SAND REPORT

SAND2004-2895
Unlimited Release
Printed June 2004

Xyce™ Parallel Electronic Simulator

Users' Guide, Version 2.0

Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Thomas V. Russo, Lon J. Waters, Eric L. Rankin, Roger P. Pawlowski, Deborah A. Fixel and Steven D. Wix

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2004-2895
Unlimited Release
Printed June 2004

Xyce™ Parallel Electronic Simulator

Users' Guide, Version 2.0

Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra,
Eric L. Rankin, Roger P. Pawlowski, Deborah A. Fixel
Computational Sciences

Thomas V. Russo, Lon J. Waters and Steven D. Wix
Component Information and Models

Sandia National Laboratories
P.O. Box 5800
Mail Stop 0316
Albuquerque, NM 87185-0316

July 22, 2004

Abstract

This manual describes the use of the **Xyce** Parallel Electronic Simulator. **Xyce** has been designed as a SPICE-compatible, high-performance analog circuit simulator, and is capabable of simulating electrical circuits at a variety of abstraction levels. Primarily, **Xyce** has been written to support the simulation needs of the Sandia National Laboratories electrical designers. This development has focused on improving capability over the current state-of-the-art in the following areas:

- Capability to solve extremely large circuit problems by supporting large-scale parallel computing platforms (up to thousands of processors). Note that this includes support for most popular parallel and serial computers.
- Improved performance for all numerical kernels (e.g., time integrator, nonlinear and linear solvers) through state-of-the-art algorithms and novel techniques.
- Device models which are specifically tailored to meet Sandia's needs, including many radiation-aware devices.

- A client-server or multi-tiered operating model wherein the numerical kernel can operate independently of the graphical user interface (GUI).
- Object-oriented code design and implementation using modern coding practices that ensure that the **Xyce** Parallel Electronic Simulator will be maintainable and extensible far into the future.

Xyce is a parallel code in the most general sense of the phrase - a message passing parallel implementation - which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved as the number of processors grows.

One feature required by designers is the ability to add device models, many specific to the needs of Sandia, to the code. To this end, the device package in the **Xyce** Parallel Electronic Simulator is designed to support a variety of device model inputs. These input formats include standard analytical models, behavioral models look-up tables, and mesh-level PDE device models. Combined with this flexible interface is an architectural design that greatly simplifies the addition of circuit models.

One of the most important feature of **Xyce** is in providing a platform for computational research and development aimed specifically at the needs of the Laboratory. With **Xyce**, Sandia now has an "in-house" capability with which both new electrical (e.g., device model development) and algorithmic (e.g., faster time-integration methods) research and development can be performed. Ultimately, these capabilities are migrated to end users.

Acknowledgements

The authors would like to acknowledge the entire Sandia National Laboratories HPEMS (High Performance Electrical Modeling and Simulation) team, including Carolyn Bogdan, Regina Schells, Ken Marx, Steve Brandon, David Shirley and Bill Ballard, for their support on this project. We also appreciate very much the work of Becky Arnold and Mike Williamson for the help in reviewing this document.

Lastly, a very special thanks to Hue Lai for typesetting this document with LaTeX.

Trademarks

Contacts

Bug Reports	`http://tvrusso.sandia.gov/bugzilla`
Email	`xyce-support@sandia.gov`
World Wide Web	`http://www.cs.sandia.gov/xyce`

This page is left intentionally blank

Contents

1. Introduction **15**
1.1 **Xyce** Overview 16
1.2 **Xyce** Capabilities 16
Support for large-scale parallel computing 16
Improved performance for all numerical kernels 17
Flexible Device Package Design 17
Modeling Fidelity 17
Analysis capability 17
Object-oriented code design and implementation 18
1.3 Description of Document 18
1.4 Reference Guide 19
1.5 How to Use this Guide 19

2. Installing and Running Xyce **21**
2.1 **Xyce** Installation 22
2.2 Running **Xyce** 23
Command Line Simulation 23
Command Line Arguments 24
Running **Xyce** in Parallel 25

3. Simulation Examples with Xyce **27**
3.1 Example Circuit Construction 28
3.2 DC Sweep Analysis 31
3.3 Transient Analysis 33

4. Netlist Basics **37**
4.1 General Overview 38
Introduction 38
Netlist Elements 38
4.2 Devices Available for Simulation 40
Analog Devices 41
4.3 Parameters and Expressions 42
Parameters 42
How to Declare and Use Parameters 42
Expressions 43

5. Working with .MODEL Statements 47
5.1 Definition of a Model 48
Defining models using model parameters 48
Defining models using subcircuit netlists 48
5.2 Model Organization 51
Model libraries 51
Model library configuration 51

6. Analog Behavioral Modeling 53
6.1 Overview of Analog Behavioral Modeling 54
6.2 Specifying ABM Devices 54
Additional constructs for use in ABM expressions 55
Lookup Tables in Analog Behavioral Modeling 55
Alternate behavioral modeling sources 56

7. Analysis Types 57
7.1 Introduction 58
7.2 DC Analysis 58
Setting Up and Running a DC Sweep 58
OP Analysis 58
7.3 Transient Analysis 61
Defining a Time-Dependent (transient) Source 61
Transient Calculation Time Steps 62
Checkpointing and Restarting 62
7.4 STEP Parametric Analysis 65
Sweeping over a Device Instance Parameter 65
Sweeping over a Device Model Parameter 67
Sweeping over Temperature 67
Special cases: Sweeping Independent Sources, Resistors, Capacitors 67

8. Using Homotopy Algorithms to Obtain Operating Points 71
8.1 Homotopy Algorithms Overview 72
HOMOTOPY Algorithms Available in **Xyce** 72
8.2 Examples 72
MOSFET Homotopy 72
Natural Parameter Homotopy 74

9. Results Output and Evaluation Options 77
9.1 Control of Results Output 78
`.PRINT` Command 78
9.2 Additional Output Options 78
`.OPTIONS OUTPUT` Command 78
9.3 Evaluating Solution Results 80

10. Guidance for Running Xyce in Parallel 81
10.1 Introduction 82

10.2 Mechanics . . . 82
10.3 Problem Size . . . 82
Smallest Possible Problem Size . . . 82
Ideal Problem Size . . . 82
10.4 Linear Solver Options . . . 83
10.5 Partitioning Options . . . 83
Chaco Static Partitioning of Circuit . . . 84
Zoltan Partitioning of the Linear System . . . 84
Singleton Filtering of the Linear System . . . 84
10.6 Recommended Partitioning and Solver Options . . . 85

11. PDE Device Simulation with Xyce **87**
11.1 Introduction . . . 88
Equations . . . 88
Discretization . . . 90
11.2 One Dimensional Example . . . 91
Netlist Explanation . . . 92
Boundary Conditions and Doping Profile . . . 93
Results . . . 93
11.3 Two Dimensional Example . . . 95
Netlist Explanation . . . 95
Doping Profile . . . 97
Boundary Conditions and Electrode Configuration . . . 97
Results . . . 97
11.4 Doping Profile . . . 102
Manually Specifying the Doping . . . 102
Default Doping Profiles . . . 105
11.5 Electrodes . . . 107
Manually Specifying the Electrodes . . . 107
Electrode Defaults . . . 110
11.6 Meshes . . . 111
Meshes from the SG Framework (External, 2D) . . . 111
Cartesian Meshes (Internal, 1D and 2D) . . . 111
Cylindrical meshes, 2D . . . 111
11.7 Mobility Models . . . 113
11.8 Bulk Materials . . . 114
11.9 Solver Options . . . 115
11.10 Output and Visualization . . . 116
Using the `.PRINT` Command . . . 116
Multi-dimensional Plots . . . 116
Volume Averaged Data . . . 117

This page is left intentionally blank

Figures

2.1 Platform scripts for running **Xyce**. 24

3.1 Schematic of diode clipper circuit with DC and transient voltage sources. . . . 29
3.2 Diode clipper circuit netlist. 30
3.3 Diode clipper circuit netlist for DC sweep analysis. 32
3.4 DC sweep voltages at Vin, node 2 and Vout. 33
3.6 Sinusoidal input signal and clipped outputs. 34
3.5 Diode clipper circuit netlist for transient analysis. 35

5.1 Example subcircuit model. 49
5.2 Example subcircuit model. 50

7.1 Diode clipper circuit netlist for DC sweep analysis. 59
7.2 DC sweep voltages at Vin, node 2 and Vout. 60
7.3 Diode clipper circuit netlist for step transient analysis. 66
7.4 Diode clipper circuit netlist for 2-step transient analysis. 68

8.1 Example MOSFET homotopy netlist. 73
8.2 Example natural parameter homotopy netlist. 75

9.1 TecPlot plot of diode clipper circuit transient response from **Xyce** `.prn` file. . . 80

11.1 MOSFET Mesh Example 89
11.2 One dimensional diode netlist. 91
11.3 Voltage regulator schematic. 92
11.4 Transient Result for voltage regulator 94
11.5 Two-dimensional BJT netlist. 96
11.6 Two-Dimensional BJT Circuit Schematic 99
11.7 Initial Two-Dimensional BJT Result 100
11.8 Final Two-Dimensional BJT Result 100
11.9 I-V Two-Dimensional BJT Result 101
11.10 One-dimensional example, with detailed doping. 103
11.11 Doping Profile 104
11.12 Two-dimensional example, with detailed doping and detailed electrodes. . . . 108
11.13 Cylindrical Mesh Example. 112

This page is left intentionally blank

Tables

1.1 **Xyce** typographical conventions. 20

2.1 List of **Xyce** command line arguments. 25

3.1 DC Analysis References 33
3.2 Transient Analysis References. 36

4.1 Analog Devices References. 40
4.2 Analog Device Quick Reference. 42
4.3 Expression operators 45
4.4 Functions in arithmetic expressions 46

7.1 Summary of time-dependent sources supported by **Xyce**. 61
7.2 Default parameters for independent sources. 69

9.1 .PRINT command options. 79

11.1 Description of the flatx, flaty doping parameters 105
11.2 Default Doping profiles for different numbers of electrodes 106
11.3 Electrode Material Options 109
11.4 Mobility models available for PDE devices 113

This page is left intentionally blank

1. Introduction

Welcome to **Xyce**

The **Xyce** Parallel Electronic Simulator has been written to support, in a rigorous manner, the simulation needs of the Sandia National Laboratories electrical designers. It is targeted specifically to run on large-scale parallel computing platforms but also runs well on a variety of architectures including single processor workstations. It also aims to support a variety of devices and models specific to Sandia needs.

1.1 **Xyce** Overview

The **Xyce** Parallel Electronic Simulator project was started in 1999 to support the simulation needs of electrical designers at Sandia National Laboratories. At this point, the current release of **Xyce** is version 2.0, and the code has evolved into a mature platform for large scale circuit simulation.

Xyce is a parallel code in the most general sense of the phrase - a message passing parallel implementation - which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows.

Xyce includes several unique features. In addition to allowing the simulation of circuits of unprecedented size, **Xyce** includes novel approaches to numerical kernels including time integration algorithms, nonlinear and linear solvers. The primary driver for this numerical innovation has been the need to simulate very large scale circuits (¿100,000 devices) on the analog level. However, it has yielded benefits, in terms of robustness and efficiency, for all classes of problems. Ideally, the increased numerical robustness minimizes the amount of simulation "tuning" required on the part of the designer.

Another feature of **Xyce** is the ability to add device models, many specific to the needs of Sandia, to the code. To this end, the device package in the **Xyce** Parallel Electronic Simulator is designed to support device model at various levels of abstraction. These include standard analytical models, behavioral models and look-up tables, and device-scale PDE models. Combined with this flexible interface is an architectural design that greatly simplifies the addition of circuit models.

1.2 **Xyce** Capabilities

Xyce has a number of unique features which are described in this section.

Support for large-scale parallel computing

Xyce is a truly parallel simulation code, designed and written from the ground up to support large-scale (up to thousands of processors) parallel computing architectures. This gives **Xyce** the capability to solve circuit problems of unprecedented size in time frames that make these simulations practical.

Xyce is a parallel code that uses a message passing parallel implementation, which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms.

Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows (*parallel scaling*).

Improved performance for all numerical kernels

In writing **Xyce** from scratch, new algorithms and heuristics have been used which improve the overall performance of the numerical kernels for a given level of accuracy. As an example, several new nonlinear solution options are available which, when coupled with iterative linear solvers, can reduce execution time for many problems.

Flexible Device Package Design

Another feature of **Xyce** is the ability to add a variety of device model types, specific to the needs of Sandia, to the code. To this end, the device package [1] in **Xyce** has been designed with a flexible device-model interface that greatly simplifies the development of circuit models. The device package interface has support for the standard "analog" or SPICE-type models, behavioral models and even PDE-based device models.

Modeling Fidelity

The requirements of Sandia National Laboratories' electrical designers include the ability to model circuit phenomena at varying levels of fidelity - even within a given simulation. While **Xyce** is primarily an analog circuit simulator, it has been designed to support a range of fidelities. To achieve this, the code has been designed with an infrastructure that supports coupled simulation at several distinct abstraction levels: device, analog, digital and mixed-signal. The code currently supports analog simulation and device-scale simulation. Development is proceeding to support digital/mixed-signal modeling, and this will be available in a later release.

Analysis capability

Xyce currently supports DC and transient as well as a variety of optimization and design options available from the DAKOTA optimization framework [1] . This document does not cover the coupling of **Xyce** with DAKOTA. Sandia customers may contact the **Xyce** team for assistance with this "developing" capability. Full support is expected in the next major release.

[1]The term "package" is a Unified Modeling Language (UML) term which refers to a group of classes, something akin to a module and is largely used in object-oriented programming.

Object-oriented code design and implementation

Xyce was designed and written from the ground up utilizing modern coding practices to ensure the optimal combination of code performance, code maintenance and code extensibility. This design allows for rapid implementation of new capability as well as long term maintenance of the code.

1.3 Description of Document

For the user, this document contains a description of the **Xyce** Parallel Electronic Simulator, in which the following topics are specifically addressed. Chapter 2 gives a quick-start guide to installing and running **Xyce**. Chapter 3 gives some examples of using the code. Chapters 4 through 7 describe the netlist language and its usage in **Xyce**. Chapter 8 covers the use of continuation algorithms, which is a unique capability for a circuit code such as **Xyce**. This is followed by Chapter 9 that covers analyzing the results output by the code. Chapter 10 provides guidance for running **Xyce** in parallel. The final chapter, Chapter 11 described the usage of mesh-based devices, which are based on solving a set of discretized partial differential equations (PDE) on a mesh, similar to commercial device simulators such as Medici.

1.4 Reference Guide

A companion document, the **Xyce** Reference Guide [2], contains more detailed information about a number of topics. Included in this document is a netlist reference for the input-file commands and elements supported within **Xyce**; a command line reference, which describes the available command line arguements for **Xyce**; and quick-references for users of other circuit codes, such as Orcad's PSpice [3] and Sandia's ChileSPICE.

1.5 How to Use this Guide

This guide is designed so you can quickly find the information you need to use **Xyce**. It assumes that you are familiar with basic Unix-type commands, how Unix manages applications and files to perform routine tasks (e.g., starting applications, opening files and saving your work).

Typographical conventions

Before continuing in this Users' Guide, it is important to understand the terms and typographical conventions used. Procedures for performing an operation are generally numbered with the following typographical conventions.

Notation	Example	Description
`Verbatim text`	`xmpirun -np 4`	Commands entered from the keyboard on the command line or text entered in a netlist.
Bold Roman Font	Set nominal temperature using the **TNOM** option.	SPICE-type parameters used in models, etc.
Gray Shaded Text	DEBUGLEVEL	Feature that is designed primarily for use by **Xyce** developers.
`[text in brackets]`	`Xyce [options] <netlist>`	Optional parameters.
`<text in angle brackets>`	`Xyce [options] <netlist>`	Parameters to be inserted by the user.
`<object with asterisk>*`	`K1 <ind. 1> [<ind. n>*]`	Parameter that may be multiply specified.

`<TEXT1|TEXT2>`	`.PRINT TRAN`
	`+ DELIMITER=<TAB|COMMA>`
		Parameters that may only take specified values.

Table 1.1. Xyce typographical conventions.

2. Installing and Running **Xyce**

Chapter Overview

This chapter describes the basic mechanics of installing and running **Xyce**. It includes the following sections:

- Section 2.1, **Xyce** *Installation*
- Section 2.2, *Running* **Xyce**

2.1 **Xyce** Installation

To obtain a copy of **Xyce**, contact the **Xyce** development team at `http://www.cs.sandia.gov/xyce`. Once you have the distribution file, install **Xyce** from the command line by following the instructions below. Examples are given for reference.

Instructions	**Examples**
Installation packages are named according the target operating system and architecture (parallel or serial).	`Install_Xyce_linux.tar.gz` (Linux Serial) `Install_Xyce_linux_MPI.tar.gz` (Linux Parallel) `Install_Xyce_windows.zip` (Windows)

Unpack the appropriate package for your platform. A similarly named installation directory is then created. Windows users can unpack with programs such as *WinZip*, *PKZip*, *Winrar*, etc.	`$ gzip -d Install_Xyce_linux.tar.gz`
	`$ tar xf Install_Xyce_linux.tar`

Enter this directory and run the installation shell script. Windows users should run the `install.bat` batch file.	`$ cd Install_Xyce_linux`
	`$ sh install_linux.sh`

Provide the requested information.	`Where should Xyce be installed?`
	` /usr/local/Xyce-2.0`

Completing the steps above will unpack **Xyce** to the specified directory. **IMPORTANT NOTE: if installing *both* serial and parallel versions of Xyce, you must specify different directories for each installation location. Failure to use different directories will cause the second installation to overwrite parts of the first and will likely yield an install that does not function.** Under the specified installation directories, the following subdirectories will be created:

- **bin** contains the executable used to start **Xyce**. The executable name will vary depending on the target operating system and architecture.
  - `runxyce` is the shell script for starting serial **Xyce** on Unix platforms.
  - `runxyce.bat` is the batch file for starting serial **Xyce** on Windows.
  - `xmpirun` is the wrapper script for mpirun used for running **Xyce** in parallel mode.
- **doc** contains the **Xyce** Users' Guide, comprehensive Reference Guide, and Release Notes. Read these for more information about this release and for detailed instructions on how to use **Xyce**.
- **lib** contains configuration files, libraries, and metadata for **Xyce**.

- **test** contains sample netlists and verification tools.

2.2 Running **Xyce**

ChileCAD [4], a GUI for **Xyce** is under development and will be part of the overall simulation framework by autumn of 2004. This manual only describes how **Xyce** is run from the command line. This section outlines how to run **Xyce** for both serial and MPI parallel simulations.

Command Line Simulation

Running **Xyce** from the command line is straightforward. The scripts `xmpirun` and `runxyce` created during installation (see Section 2.1) set up the runtime environment and execute **Xyce**. (Microsoft Windows users should launch the "Command Prompt" window and use the `runxyce.bat` batch file.) Depending on whether you are using a version compiled with MPI support or a serial version, there are two ways to begin running **Xyce**:

- Running serial **Xyce**:

  ```
  > runxyce [options] <netlist filename>
  ```

- Running **Xyce** in parallel:

  ```
  > xmpirun -np <# procs> [options] <netlist filename>
  ```

where [`options`] are the command line arguments for **Xyce**. For example, to log output to a file named `sample.log` type:

```
$ runxyce -l sample.log <netlist filename>
```

The next example runs parallel **Xyce** on four processors and places the resultsinto a comma separated value file named `results.csv`:

```
$ xmpirun -np 4 -delim COMMA -o results.csv <netlist filename>
```

These examples assume that `<netlist filename>` is either in the current working directory or includes the path (full or relative) to the netlist file. Enclose the filename in quotation marks (" ") if the path contains spaces. Help is accessible with the `-h` option.

For MPI runs, [`options`] may also include command line arguments to `mpirun`. Consult the documentation installed with MPI on your platform for more details on MPI options. The

`-np <# procs>` denotes the number of processors to use for the simulation. *NOTE: It is critical that the number of processors used is less than the number of devices and voltage nodes in the netlist.* The appropriate script used to run **Xyce** for each supported platform is listed in the Table 2.1.

Computer Architecture	OS	Serial Executable	MPI Executable
Apple PPC	OSX	`runxyce`	Not Available
HP/Compaq	Tru64	`runxyce`	`xmpirun`
SGI 64 bit	IRIX 6.5	`runxyce`	`xmpirun`
Intel X86	Linux	`runxyce`	`xmpirun`
Intel X86	FreeBSD	`runxyce`	`xmpirun`
Intel X86	Microsoft Windows 2000	`runxyce.bat`	Not Available

Figure 2.1. Platform scripts for running **Xyce**.

While **Xyce** is running, the progress of the simulation is output to the command line window. See the **Xyce** Reference Guide for complete list and explanation of command line options.

Command Line Arguments

Xyce supports a handful of command line arguments which must be given *before* the netlist filename. While most of these are intended for general use, others simply give access to new features that, while supported, are not enabled by default. These options are designated as *trial* options. The general usage is as follows:

```
runxyce [arguments] <netlist filename>
```

Table 2.1 gives a complete lists of command line options. In this table, the shaded rows indicate the trial options. *DEPRECATED* options are no longer supported and will be removed from future releases.

Argument	Description	Usage	Default
-h	Help option. Prints usage and exits.	`-h`	-
-v	Prints the version banner and exits.	`-v`	-

Argument	Description	Usage	Default
-op	DEPRECATED. Use the old version of the netlist parser.	`-op`	-
-delim	Set the output file field.	`-delim <TAB|COMMA|string>`	-
-o	Place the results into specified file.	`-o <file>`	-
-l	Place the log output into specified file.	`-l <file>`	-
-nox	Use the NOX nonlinear solver.	`-nox <on|off>`	on
-dva	Use faster direct vector access.	`-dva <on|off>`	on
-dma	Use faster direct matrix access.	`-dma <on|off>`	on

Table 2.1: List of **Xyce** command line arguments.

Running **Xyce** in Parallel

A parallel version of **Xyce** is available for several different platforms as shown in Table 2.1. Running **Xyce** in parallel requires that the correct version of `mpirun` is used. Use the script `xmpirun` to call the correct version with the appropriate parameters. For example, to run **Xyce** on two processors with an example netlist, type:

```
xmpirun -np 2 anExampleNetlist.cir
```

In general the number of processors is specified by using the `-np` argument to the appropriate `mpirun` command. Some specific considerations are given below.

Running Xyce under MPICH

The MPICH implementation of MPI requires that there exist a file of machines on which to run. On RedHat Linux this is installed in /usr/lib/mpich/share. On FreeBSD this is installed in /usr/local/mpich/share. This file must contain one line for each machine on which a process may be started. If you do not have write access to the directory in which the default machines file is stored you may specify an alternate file with the `-machinefile <machinefilename>` option to `mpirun`.

MPICH executes parallel jobs by using the remote shell (rsh) or secure shell (ssh) to the target machine. You might, therefore, be prompted for a password when starting up a multiple processor job.

Running Xyce under LAM MPI

Unlike MPICH, LAM MPI requires a daemon process to be running on each machine that will service parallel jobs. This daemon is started by using the `lamboot` program. By default, `lamboot` will run a daemon on the local machine, but it may be given a file name containing a list of machines for multiple-machine jobs. Consult the `bhost` man page for the format of the file.

`lamboot` runs a program called `lamd` which will remain running until it is halted. As long as `lamd` is running you may continue to run parallel jobs. Halt `lamd` using the `lamhalt` command.

Guidance

This chapter has given the basic mechanics of running **Xyce** in parallel. For general guidance regarding solver options, partioning options, and other parallel issues, refer to chapter 10. Distributed memory circuit simulation still contains a number of research issues, so obtaining an optimal simulation in parallel is a bit of an art.

3. Simulation Examples with **Xyce**

Chapter Overview

This chapter provides several simple examples of **Xyce** usage. An example circuit is provided for each available analysis type.

- Section 3.1, *Example Circuit Construction*
- Section 3.2, *DC Sweep Analysis*
- Section 3.3, *Transient Analysis*

3.1 Example Circuit Construction

This section describes how to use **Xyce** to create the simple diode clipper circuit shown in Figure 3.1.

While a schematic edit and capture capability is under development, **Xyce** currently only supports circuit creation via netlist editing. **Xyce** supports most of the standard netlist entries common to Berkeley SPICE 3F5 and Orcad PSpice. For users who are familiar with PSpice netlists, the differences between PSpice and **Xyce** netlists are listed in the **Xyce** Reference Guide [2].

Example: diode clipper circuit

1. Open a new netlist file using a standard text editor (e.g., VI, Emacs, notepad, etc.).
2. Type the title on the first line of the netlist:

   ```
   Diode Clipper Circuit
   ```

3. Create a 5V DC voltage source between nodes 1 and 0 by typing the following on a new line:

   ```
   VCC 1 0 5V
   ```

4. Create another DC voltage source between nodes 3 and 0 by entering the following on a new line:

   ```
   VIN 3 0 0V
   ```

5. Place the diodes in the circuit between nodes 2 and 1, and nodes 0 and 2, respectively, by entering the following lines:

   ```
   D1 2 1 D1N3940
   D2 0 2 D1N3940
   ```

6. Enter resistors R1, R2, R3 and R4, respectively:

   ```
   R1 2 3 1K
   R2 1 2 3.3K
   R3 2 0 3.3K
   R4 4 0 5.6K
   ```

7. Place the capacitor in the circuit:

```
C1 2 4 0.47u
```

8. Add the diode model to the netlist to complete it as Figure 3.2.
9. Complete the netlist by entering `.END` on the last line in the file. Save the file as `clipper.cir`. The complete netlist is shown in Figure 3.2 and the schematic in Figure 3.1.

The netlist in Figure 3.2 illustrates some of the syntax of a netlist input file. Netlists begin with a title (e.g., "`Diode Clipper Circuit`"), support comments (lines beginning with the "*" character), devices, model definitions and the "`.END`" statement.

This netlist file is not yet complete and will not run properly using **Xyce** (see Section 2.2 for instructions on running **Xyce**) as it lacks an analysis statement. As you proceed in this chapter, you will see how to add the appropriate analysis statement and run the clipper circuit.

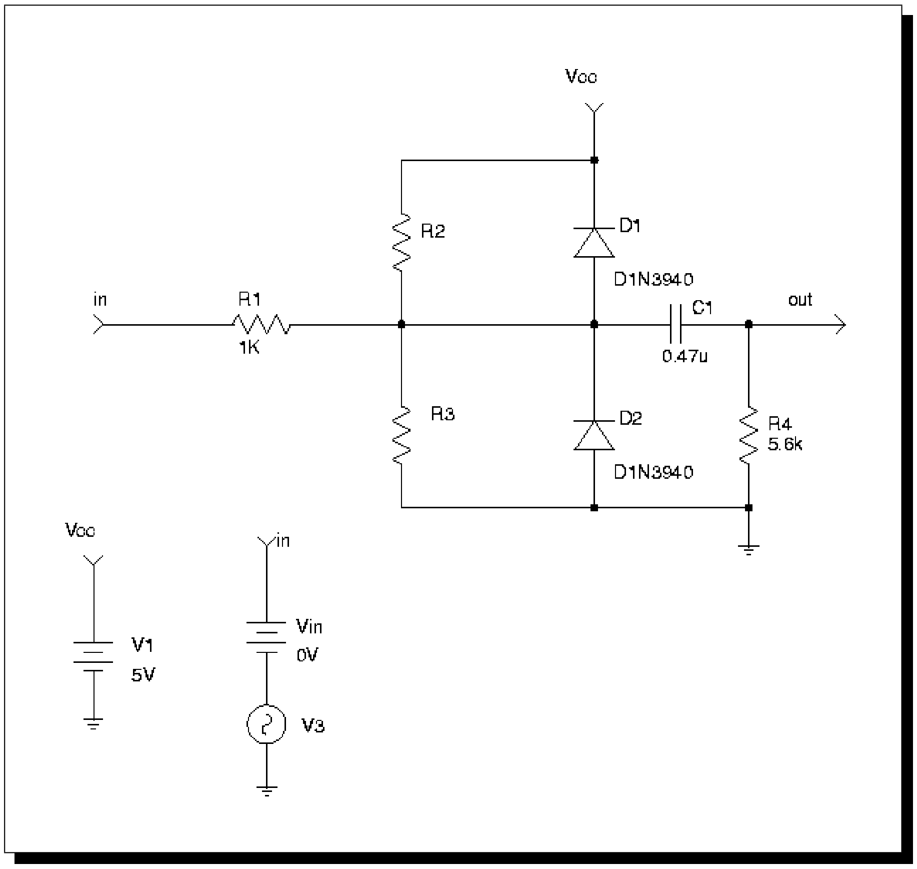


Figure 3.1. Schematic of diode clipper circuit with DC and transient voltage sources.

```
Diode Clipper Circuit
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3.2. Diode clipper circuit netlist.

3.2 DC Sweep Analysis

This section illustrates how to run a DC sweep analysis using **Xyce**. In this example we examine the DC response of the clipper circuit by running a DC sweep of the input voltage source (`Vin`) and reviewing at the results generated by **Xyce**. This example demonstrates using DC sweep analysis parameters that vary `Vin` from -10 to 15 volts in 1 volt steps.

Example: DC sweep analysis

To set up and run a DC sweep analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file (`clipper.cir`) using a standard text editor (e.g., VI, Emacs, notepad, etc.).

2. Enter the analysis control statement in the netlist:

   ```
   .DC VIN -10 15 1
   ```

3. Enter the output control statement:

   ```
   .PRINT DC V(3) V(2) V(4)
   ```

4. Save the netlist file and run **Xyce** on the circuit. For example, to run serial **Xyce**:

   ```
   > runxyce clipper.cir
   ```

5. Open the results file (clipper.cir.prn) and examine (or plot) the output voltages that were selected for nodes 3 (Vin), 2 and 4 (Out). Figure 3.4 shows the output plotted as a function of the swept variable Vin.

The modified netlist is shown below in Figure 3.3.

```
Diode Clipper Circuit with DC sweep analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+         IS = 4E-10
+         RS = .105
+          N = 1.48
+         TT = 8E-7
+        CJO = 1.95E-11
+         VJ = .4
+          M = .38
+         EG = 1.36
+        XTI = -8
+         KF = 0
+         AF = 1
+         FC = .9
+         BV = 600
+        IBV = 1E-4)
*
.END
```

Figure 3.3. Diode clipper circuit netlist for DC sweep analysis.

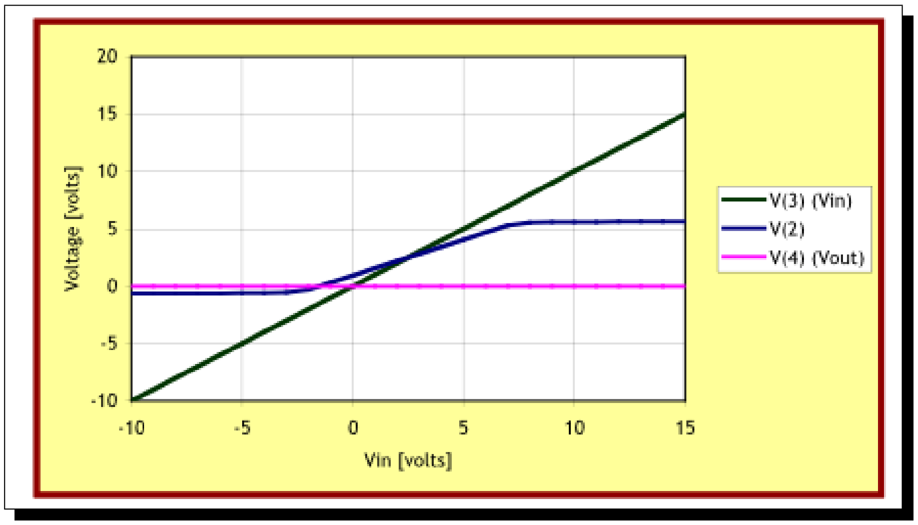


Figure 3.4. DC sweep voltages at Vin, node 2 and Vout.

Table 3.1 gives references for further explanation of the supported DC sweep analysis.

To find out more about...	See...
DC analysis for analog designs	Chapter 7.2, DC Analysis

Table 3.1. DC Analysis References

3.3 Transient Analysis

This section shows how to run a transient analysis using **Xyce**. In this example, we look at the transient response of the clipper circuit to a sinusoidal input voltage source (`Vin`) and review the results generated by **Xyce**. This example utilizes a sinusoidal input voltage source running at a frequency of 1 kHz and amplitude of 10 volts. To set up this example, we must modify the netlist to include this source.

Example: transient analysis

To set up and run a transient analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file file (clipper.cir) using a standard text editor (e.g., VI, Emacs, notepad, etc.).

2. If you added DC analysis statements in the previous example, remove them (see Figure 3.4).
3. Enter the analysis control in the netlist:

```
.TRAN 2ns 2ms
```

4. Modify the input voltage source (`Vin`) to generate the sinusoidal input signal:

```
VIN 3 0 SIN(10V 1kHz)
```

5. Save the netlist file and run **Xyce** on the circuit. For example, to run serial **Xyce**:

```
> runxyce clipper.cir
```

6. Open the results file and examine (or plot) the output voltages for nodes 3 (`Vin`), 2 and 4 (`Out`). The plot in Figure 3.6 shows the output plotted as a function of time.

The modified netlist is shown in Figure 3.5.

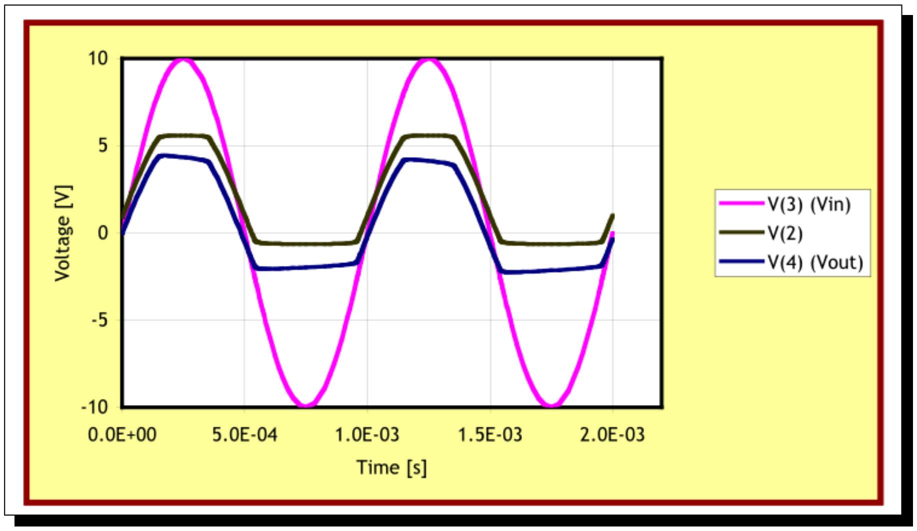


Figure 3.6. Sinusoidal input signal and clipped outputs.

Table 3.2 below gives references for further explanation of the supported transient analysis.

```
Diode Clipper Circuit with transient analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+         IS = 4E-10
+         RS = .105
+          N = 1.48
+         TT = 8E-7
+        CJO = 1.95E-11
+         VJ = .4
+          M = .38
+         EG = 1.36
+        XTI = -8
+         KF = 0
+         AF = 1
+         FC = .9
+         BV = 600
+        IBV = 1E-4)
*
.END
```

Figure 3.5. Diode clipper circuit netlist for transient analysis.

To find out more about...	**See...**
Transient analysis for analog designs	Chapter 7.3, Transient Analysis

Table 3.2. Transient Analysis References.

4. Netlist Basics

Chapter Overview

This chapter contains introductory material on netlist syntax and usage. Sections include:

- Section 4.1 *General Overview*
- Section 4.2 *Devices Available for Simulation*
- Section 4.3 *Parameters and Expressions*

4.1 General Overview

Introduction

Using a netlist to describe a circuit for **Xyce** is the primary method used for running a circuit simulation. Netlist support within **Xyce** largely conforms to that used by Berkeley SPICE 3F5 with several new options for controlling functionality unique to **Xyce**. In a netlist, the circuit is described by a set of "element lines" which define the circuit elements and their values, the circuit topology (the connection of the circuit elements), and a variety of control options for the simulation. The first line in the netlist file must be a title and the last line must be "`.END`". Between these two constraints, the order of the statements is irrelevant.

Netlist Elements

An "element line", for which the format is determined by the specific element type, defines each circuit element instance. The general format is given by:

```
<type><name> <node information> <element information...>
```

The `<type>` must be a letter (A through Z) and the `<name>` follows immediately. For example, `RARESITOR` specifies a type=resistor with a name `ARESITOR`. Fields on a line are separated by spaces, commas, an equal sign or a left or right parenthesis.

A number field may be an integer or a floating-point value. Either one may be followed by one of the following scaling factors:

Symbol	Equivalent Value
`T`	10^{12}
`G`	10^{9}
`Meg`	10^{6}
`K`	10^{3}
`mil`	25.4^{-6}
`m`	10^{-3}
`u` (μ)	10^{-6}
`n`	10^{-9}
`p`	10^{-12}
`f`	10^{-15}

Node information is given in terms of node names, which are arbitrary character strings. The only requirement is that the ground node is named '`0`'. There are some restrictions on the circuit topology:

- There can be no loop of voltage sources and/or inductors.
- There can be no cut-set of current sources and/or capacitors.
- Every node must have a DC path to ground.
- Every node must have at least two connections (with the exception of unterminated transmission lines and MOSFET substrate nodes).

The following line provides an example of an element line that defines a resistor between nodes `1` and `3` with a resistance value of $10\text{k}\Omega$.

Example: `RARESISTOR 1 3 10K`

Title, Comments and End

The first line of the netlist is the title line of the netlist and is included in the output file. It is a common mistake to forget the meaning of this first line and begin the circuit elements on the first line; doing so will probably result in a parsing error.

Example: `Test RLC Circuit`

The "`.End`" line must be the last line in the netlist.

Example: `.END`

Comments are supported in netlists and are indicated by placing an asterisk at the beginning of the comment line. They may occur anywhere in the netlist *but* they must be at the beginning of a line. **Xyce** also supports "in-line" comments. An in-line comment is designated by a semicolon and may occur on any line. Everything after the semicolon is taken as a comment and ignored. Any line that begins with leading white space is also considered to be a comment.

Example: `* This is a netlist comment.`

Example: **WRONG:**`.DC ....  * This type of inline comment is` *`not supported.`*

Example: `.DC ....  ; This type of inline comment is supported.`

Netlist Commands

Command elements are used to describe the analysis being defined by the netlist. Examples include analysis types, initial conditions, device models and output control. The **Xyce** Reference Guide [2] contains a reference for these commands.

Example: `.PRINT TRAN V(Vout)`

Analog Devices

The analog devices supported include most of the standard circuit components normally found in circuit simulators such as SPICE 3F5, PSpice, etc., plus several Sandia specific devices.

Example: `D_CR303 N_0065 0 D159700`

Table 4.1 below gives references for further explanation of the supported analog devices.

To find out more about...	See...
Analog devices	**Xyce** Reference Guide [2]

Table 4.1. Analog Devices References.

4.2 Devices Available for Simulation

This section describes the different types of analog devices supported in **Xyce**. These include standard analog devices, sources (dependent and independent) and subcircuits. Each device description has the following information:

- A description and an example of the netlist syntax
- The corresponding model types and descriptions, where applicable
- The corresponding lists of model parameters and descriptions, where applicable
- The associated circuit diagram and model equations (as necessary)

These analog devices include all of the standard circuit components needed for most analog circuits. User defined models may also be implemented using the `.MODEL` (model definition) statement and macromodels as subcircuits using the `.SUBCKT` (subcircuit) statement.

Analog Devices

Xyce supports many analog devices, including sources, subcircuits and behavioral models. The devices are classified into device types, each of which can have one or more model types. For example, the BJT device type has two model types: NPN and PNP.

The device element statements in the netlist always start with the name of the individual device instance. The first letter of the name determines the device type. The format of the following information depends on the device type and its parameters. The Device Type summary table (Table 4.2) lists all of the analog devices supported by **Xyce**. Each standard device is then described in more detail in the following sections. Except where noted, the devices are based upon those found in [5].

Table 4.2 is a summary of the analog device types and the form of their netlist formats. For a more complete description of the syntax for supported devices, see the **Xyce** Reference Guide. [2].

Device Type	Designator Letter	Typical Netlist Format
Capacitor	`C`	`C<name> <+ node> <- node> [model name] <value>` `+ [IC=<initial value>]`
Inductor	`L`	`L<name> <+ node> <- node> [model name] <value>` `+ [IC=<initial value>]`
Resistor	`R`	`R<name> <+ node> <- node> [model name] <value>` `+  [L=<length>] [W=<width>]`
Diode	`D`	`D<name> <anode node> <cathode node>` `+ <model name> [area value]`
Mutual Inductor	`K`	`K<name> <inductor 1> [<ind. n>*]` `+ <linear coupling or model>`
Independent Voltage Source	`V`	`V<name> <+ node> <- node> [[DC] <value>]` `+  [transient specification]`
Independent Current Source	`I`	`I<name> <+ node> <- node> [[DC] <value>]` `+  [transient specification]`
Voltage Controlled Voltage Source	`E`	`E<name> <+ node> <- node> <+ controlling node>` `+ <- controlling node> <gain>`
Voltage Controlled Current Source	`G`	`G<name> <+ node> <- node> <+ controlling node>` `+ <- controlling node> <transconductance>`
Nonlinear Dependent Source (B Source)	`B`	`B<name> <+ node> <- node>` `+ <I or V>={<expression>}`
Bipolar Junction Transistor (BJT)	`Q`	`Q<name> <collector node> <base node> <emitter node> [substrate node] <model name> [area value]`

Device Type	Designator Letter	Typical Netlist Format
MOSFET	M	`M<name> <drain node> <gate node> <source node>` `+ <bulk/substrate node> <model name>` `+ [common model parameter]*`
Transmission Line	T	`T<name> <A port + node> <A port - node>` `+ <B port + node> <B port - node>` `+ <ideal specification>`
Voltage Controlled Switch	S	`S<name> <+ switch node> <- switch node>` `+ <+ controlling node> <- controlling node>` `+ <model name>`
Subcircuit	X	`X<name> [node]* <subcircuit name>` `+ [PARAMS:[<name>=<value>]*]`
PDE Devices	Z	`Z<name> <node1> <node2> [node3]` `+ [node4] <model name>`

Table 4.2: Analog Device Quick Reference.

4.3 Parameters and Expressions

In addition to explicit values, the user may use parameters and expressions to symbolize numeric values in the circuit design.

Parameters

A parameter is like a programming variable that represents a numeric value by name. Once you have defined a parameter (declared its name and given it a value) at a particular level in the circuit hierarchy, you can use it to represent circuit values at that level or any level directly beneath it in the circuit hierarchy. One way that you can use parameters is to apply the same value to multiple part instances.

How to Declare and Use Parameters

In order to use a global parameter in a circuit, one must:

- define the parameter using a `.PARAM` statement within a netlist
- replace an explicit value with the parameter in the circuit

Note that **Xyce** reserves several keywords that may not be used as parameter names. These are:

- `Time`
- `Vt`
- `Temp`
- `GMIN`

However, in this release of **Xyce**, only `Time` is predefined.

Example: Declaring a parameter

1. Locate the level in the circuit hierarchy at which the `.PARAM` statement declaring a parameter will be placed (note: a global parameter that can be used anywhere in the netlist can be declared by placing the `.PARAM` statement at the top-most level of the circuit).
2. Name the parameter and give it a value. The value can be numeric or given by an expression:

   ```
   .SUBCKT subckt1 n1 n2 n3
   .PARAM res = 100
   *
   * other netlist statements here
   *
   .ENDS
   ```
3. Note: the parameter "res" can be used anywhere within the subcircuit subckt1 including subcircuits defined within it, but cannot be used outside of subckt1.

Example: Using a parameter in the circuit

1. Find the numeric value that is to be replaced by a parameter: a device instance parameter value, model parameter value, etc. The value being replaced must be accessible with the current hierarchy level.
2. Replace the numeric value with the parameter name contained within braces ({}) as in:

   ```
   R1 1 2 {res}
   ```

Expressions

In **Xyce**, an expression is a mathematical relationship that may be used any place one would use a number (numeric or boolean). Except in the case of expressions used in analog behavioral modeling sources (see Chapter 6) **Xyce** evaluates the expression to a value when it reads in the circuit netlist, not each time its value is needed. It is therefore necessary that all terms in an expression be known at the beginning of the run.

To use an expression in a circuit netlist:

1. Locate the value to be replaced (component, model parameter, etc.).
2. Substitute the value with an expression utilizing the {} syntax:

   {*expression*}

   where *expression* can contain any of the following:

   - available operators from those in Table 4.3
   - included functions from those in Table 4.4
   - user-defined functions
   - user-defined parameters that are within scope
   - literal operands

   The braces ({}) instruct **Xyce** to evaluate the expression and use the resulting value.

   Additional time-dependent constructs are available in expressions used in analog behavioral modeling sources (see Chapter 6).

Example: Using an expression

Scaling the DC voltage of a $12V$ independent voltage source, designated `VF`, by some factor can be accomlished by the following netlist statements (in this example the factor is 1.5):

```
.PARAM FACTORV=1.5
VF 3 4 {FACTORV*12}
```

Xyce will evaluate the expression to:

`12 * 1.5` or `18` volts

[1]Logical and relational operators are used only with the `IF()` function.

Class of operator...	Operator...	Meaning
arithmetic	`+`	addition or string concatenation
	`-`	subtraction
	`*`	multiplication
	`/`	division
	`**`	exponentiation
logical[1]	`~`	unary NOT
	`|`	boolean OR
	`^`	boolean XOR
	`&`	boolean AND
relational	`==`	equality
	`!=`	non-equality
	`>`	greater-than
	`>=`	greater-than or equal
	`<`	less-than
	`<=`	less-than or equal

Table 4.3. Expression operators

Function...	Meaning...	Explanation...
ABS(x)	$\|x\|$	
SQRT(x)	$\sqrt{x}$	
MIN(x,y)	$\min(x, y)$	minimum of x and y
MAX(x,y)	$\max(x, y)$	maximum of x and y
EXP(x)	e^x	
LN(x)	$\ln(x)$	log base e
LOG(x)	$\log(x)$	log base 10
SIN(x)	$\sin(x)$	x in radians
ASIN(x)	$\arcsin(x)$	result in radians
SINH(x)	$\sinh(x)$	x in radians
ASINH(x)	$\sinh^{-1}(x)$	result in radians
COS(x)	$\cos(x)$	x in radians
ACOS(x)	$\arccos(x)$	result in radians
COSH(x)	$\cosh(x)$	x in radians
ACOSH(x)	$\cosh^{-1}(x)$	result in radians
TAN(x)	$\tan(x)$	x in radians
ATAN(x)	$\arctan(x)$	result in radians
TANH(x)	$\tanh(x)$	x in radians
ATANH(x)	$\tanh^{-1}(x)$	result in radians
ATAN2(x,y)	$\arctan(y/x)$	result in radians
SGN(x)	+1 if $x > 0$ 0 if $x = 0$ -1 if $x < 0$	
STP(x)	1 if $x > 0$ 0 otherwise	suppress a value until a given time
URAMP(x)	x if $x > 0$ 0 otherwise	
IF(t,x,y)	x if t is true, y otherwise	t is an expression using the relational operators in Table 4.3
DDT(x)	time derivative of x	
SDT(x)	time integral of x	

Table 4.4. Functions in arithmetic expressions

5. Working with .MODEL Statements

Chapter Overview

This chapter contains model ideas and a summary of the ways to create and modify models. Sections include:

- Section 5.1, *Definition of a Model*
- Section 5.2, *Model Organization*

5.1 Definition of a Model

A model describes the electrical performance of a *part*. A part is a component in the circuit with *specific simulation properties* that *define* the part. In a netlist, a part is identified by its implementation properties designated by the associated model name.

Depending on the given device type, a model is defined as either:

- a model parameter set
- a subcircuit netlist

Both methods of defining a model use a netlist format, with precise syntax rules as described below.

Defining models using model parameters

Xyce currently has no built-in models. However, models can be defined for a device by changing some or all of the *model parameters* from their defaults via the `.MODEL` syntax. For example:

```
M5 3 2 1 0 MLOAD1
.MODEL MLOAD1 NMOS (LEVEL=3 VTO=0.5 CJ=0.025pF)
```

This example defines a MOSFET device, then specifies parameters defining that device in the `.MODEL` line.

Most device types in **Xyce** support some form of model parameters. Consult the **Xyce** Reference Guide [2] for the model parameters supported by each device type.

Defining models using subcircuit netlists

In **Xyce**, models may also be defined using the *subcircuit syntax:* `.SUBCKT`/`.ENDS`. This syntax includes:

- *netlists* to define the configuration and function of the part.
- *variable input parameters* to modify the model.

See Figure 5.1 for an example.

```
****other devices
X5 5  6  7  8 l3dsc1 PARAMS: ScaleFac=2.0
X6 9 10 11 12 l3dsc1
****more netlist commands

*** SUBCIRCUIT: l3dsc1
*** Parasitic Model: microstrip
*** Only one segment
.SUBCKT l3dsc1 1 3 2 4 PARAMS: ScaleFac=1.0
C01 1 0 4.540e-12
RG01 1 0 7.816e+03
L1 1 5 3.718e-08
R1 5 2 4.300e-01
C1 2 0 4.540e-12
RG1 2 0 7.816e+03
C02 3 0 4.540e-12
RG02 3 0 7.816e+03
L2 3 6 3.668e-08
R2 6 4 4.184e-01
C2 4 0 4.540e-12
RG2 4 0 7.816e+03
CM012 1 3 5.288e-13
KM12 L1 L2 2.229e-01
CM12 2 4 {5.288e-13*ScaleFac}
.ENDS
```

Figure 5.1. Example subcircuit model.

In this example, a subcircuit model called `l3dsc1` implementing one part of a microstrip transmission line is defined between the `.SUBCKT`/`.ENDS` lines, and two diferent instances of the subcircuit are used in the `X` lines. This somewhat artificial example shows how input parameters are used; the last capacitor in the subcircuit is scaled by the input parameter `ScaleFac`. If input parameters are not specified on the `X` line (as in the case of device `X6`), then the default values specified on the `.SUBCKT` line are used. Non-default values are specified on the `X` line using the `PARAMS:` keyword. For precise syntax consult the **Xyce** Reference Guide [2].

Subcircuit Hierarchy

Xyce supports the definition of subcircuits within other subcircuits. Each subcircuit definition introduces a new level in the circuit hierarchy with the top level being the main circuit. If a second level is defined, it is composed of the subcircuits in the main circuit and each subsequent level is composed of the subcircuits contained in the previous level. A subcircuit may also contain other definitions such as models via the `.MODEL` statement, parameters via the `.PARAM` statement, and functions via the `.FUNC` statement.

In this context, the subcircuit defines the "scope" for the definitions it contains. That is, *the definitions contained within a subcircuit can be used within that subcircuit and/or within any subcircuit it contains*. Any definitions occuring in the main circuit have global scope and can be used anywhere in the circuit. A name, such as a model, parameter, function or subcircuit name, occurring in a definition at one level of a circuit hierarchy can be redefined at any lower level contained directly by the subcircuit. In this case, the new definition applies at the given level and those below.

In the following example, the model named `MOD1` can be used in subcircuits `SUB1` and `SUB2` but not in the subcircuit `SUB3`. The parameter `P1` has a value of 10 in subcircuit `SUB1` and a value of 20 in subcircuit `SUB2`.

```
.SUBCKT SUB1 1 2 3 4
.MODEL MOD1 NMOS(LEVEL=2)
.PARAM P1=10
*
* subcircuit devices omitted for brevity
*
.SUBCKT SUB2 1 3 2 4
.PARAM P1=20
*
* subcircuit devices omitted for brevity
*
.ENDS
.ENDS

.SUBCKT SUB3 1 2 3 4
*
* subcircuit devices omitted for brevity
*
.ENDS
```

Figure 5.2. Example subcircuit model.

5.2 Model Organization

While it is always possible to make a self-contained netlist in which all models for all parts are include along with the circuit definition, it is often more convenient to organize frequently-used models into separate model libraries. **Xyce** provides a very simple mechanism that allows this organization. Models are simply collected into model library files, and then accessed by netlists as needed by insertion of an `.INCLUDE` directive. This section describes the process in detail.

Model libraries

Device model and subcircuit definitions may be organized into model libraries. These libraries are text files (similar to netlist files) that have one or more model definitions. Model library names usually end with a `.lib` extension, but may be named using any convention the user chooses.

As a rule-of-thumb, model libraries files typically include similar model types. In these files, the *header comments* describe the models therein.

Model library configuration

In **Xyce**, model libraries are implemented using the `.INCLUDE` statement. Once a file is included, its contents are available to the netlist just as if the entire contents had been inserted directly into the netlist.

As an example, one might create the following model library file called `bjtmodels.lib`, containing `.MODEL` statements for common types of bipolar junction transistors:

```
*bjtmodels.lib
* Bipolar transistor models
.MODEL Q2N2222 NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=5 Ne=1.307
+  Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+  Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+  Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)

.MODEL 2N3700 NPN (IS=17.2E-15 BF=100)

.MODEL 2N2907A PNP (IS=1.E-12 BF=100)
```

The models `Q2N2222`, `2N3700` and `2N2907A` could then be used in a netlist by including the `bjtmodels.lib` file.

```
.INCLUDE "bjtmodels.lib"
Q1 1 2 3 Q2N2222
Q2 5 6 7 2N3700
Q3 8 9 10 2N2907A
*other netlist entries
.END
```

Because the contents of an included file are simply inserted into the netlist at the point where the `.INCLUDE` statement appears, the scoping rules for `.INCLUDE` statements is the same as for other types of definitions as outlined in the preceding section. Note that the path to the library file is assumed to be relative to the execution directory, but absolute pathnames are permissible.

6. Analog Behavioral Modeling

Chapter Overview

This chapter contains a description of analog behavioral modeling in **Xyce**. Sections include:

- Section 6.1,*Overview of Analog Behavioral Modeling*
- Section 6.2, *Specifying ABM Devices*

6.1 Overview of Analog Behavioral Modeling

The analog behavioral modeling capability of **Xyce** provides for flexible descriptions of electronic components in terms of a transfer function or lookup table. In other words, a mathematical relationship is used to model a circuit segment removing the need for component by component design.

The primary device used for analog behavioral modeling in **Xyce** is the `B` device, or nonlinear dependent source. A `B` device can serve as a voltage or current source, and by using expressions dependent on voltages and currents elsewhere in the circuit the user can produce any desired behavior.

6.2 Specifying ABM Devices

ABM devices (`B` devices) are specified in a netlist the same way as other devices. Customizing the operational behavior of the device is achieved by defining an ABM expression describing how inputs are transformed into outputs.

For example, the pair of lines below would provide exactly the same behavior as a 10K resistor between nodes 1 and 2. It is written to be a current source with the current specified using Ohm's law and the constant resistance.

```
.PARAM Res1=10K
Blinearres 1 2 I={(V(2)-V(1))/Res1}
```

A nonlinear resistor could be specified similarly:

```
.PARAM R1=0.15
.PARAM R2=6
.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}

.FUNC Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

Bnlr 4 2 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+ )
+ )}
```

In this example, `Bnlr` provides a voltage between nodes 4 and 2, and the voltage is determined using Ohm's law with a resistance that is a function of the voltage on node 101 and a number of parameters. These two examples demonstrate how the `B` source can be used either as a voltage source (by specifying `V={expression}`) or as a current source (with `I={expression}`).

Note that unlike expressions used in parameters or function declarations, expressions in the nonlinear dependent source may contain voltages and currents from other parts of the circuit, or even explicit time-dependent functions. These expressions are evaluated every time the current or voltage through the source are needed.

Additional constructs for use in ABM expressions

ABM expressions follow the same rules as other expressions in a netlist with the additional ability to specify signals (node voltages and voltage source currents) and explicitly time-dependent functions in the expression. In ABM expressions, refer to signals by name. **Xyce** recognizes the following constructs in ABM expressions:

- `V(<node name>)`
- `V(<node name>,<node name>)`
- `I(<voltage source name>)`
- The variable `TIME`
- Lookup tables

In a hierarchical circuit (a circuit with possibly nested levels of subcircuits), voltage source names that appear in an ABM expression must be the name of a voltage source in the same subcircuit as the ABM device. Similarly, node names in an ABM expression must be the node names of one or more devices in the same subcircuit as the ABM device.

Lookup Tables in Analog Behavioral Modeling

Lookup tables provide a means of specifying a piecewise-linear function in an expression. A table expression is specified with the keyword `TABLE` followed by an expression that is evaluated as the independent variable of the piecewise linear function, followed by a list of pairs of independent variable/dependent variable values. For example

Example: B1 1 0 V={TABLE {time} = (0, 0) (1, 2) (2, 4) (3, 6)}

will produce a voltage source whose voltage is a simple linear function of time. At $t = 0$ the voltage is 0 volts, at time $t = 1s$ the voltage is 2 volts, and at times in between the voltage is determined by linear interpolation.

The independent variable of the table source does not have to be a simple expression:

Example: B1 1 0 V={TABLE {V(5)-V(3)/4+I(V6)*Res1} = (0, 0) (1, 2) (2, 4) (3, 6)}

Alternate behavioral modeling sources

In addition to the primary nonlinear dependent source, the `B` source, **Xyce** also supports the PSpice extensions to the standard Spice voltage- and current-controlled sources, the `E`, `F`, `G` and `H` sources. These sources are provided for PSpice compatibility, and are converted internally into `B` sources. See the Netlist Reference chapter of the **Xyce** Reference Guide [2] for the syntax of these compatibility devices.

7. Analysis Types

Chapter Overview

This chapter contains a description of the different analysis types available in **Xyce**. It includes the following sections:

- Section 7.2, *DC Analysis*
- Section 7.3, *Transient Analysis*
- Section 7.4, *STEP Parametric Analysis*

7.1 Introduction

Several simulation analysis options are supported within **Xyce**. For basic analysis, **Xyce** currently supports DC and transient analysis; AC analysis is intended to be supported in a future release. STEP parametric analysis, which applies an outter parameter loop to either DC or transient analysis is available. Also, a variety of optimization and design options are available via coupling with the DAKOTA optimization framework [1]. While DAKOTA is not distributed as part of **Xyce**, Sandia customers may contact the **Xyce** team for assistance with this capability.

7.2 DC Analysis

The DC sweep analysis capability in **Xyce** carries out a sweep, in DC mode, on a circuit. DC sweep is supported for a source (current or voltage), through a range of specified values. As the sweep proceeds, the bias point is computed for each value in the specified range of the sweep.

If the variable to be swept is a voltage or current source, a DC source must be used. The DC value is set in the netlist (see the **Xyce** Reference Guide [2]). In simulating the DC response of an analog circuit, **Xyce** eliminates any time dependence from the circuit. This is accomplished by treating all capacitor elements as open circuits, all inductor elements as short circuits and using only the DC values of both voltage and current sources.

Setting Up and Running a DC Sweep

Following the example given in Section 3.2, the diode clipper circuit netlist is shown in Figure 7.1 with a DC sweep analysis specified. Here, the voltage source `Vin` is swept from -10 to 15 in 1 volt increments, resulting in 26 DC operating point calculations. Note also that the default setting for `Vin` is ignored during these calculations. All other source values use the specified values (`VCC = 5V` in this case).

Running **Xyce** on this netlist produces an output results file named `clipper.cir.prn`. Plotting this data produces the graph shown in Figure 7.2.

OP Analysis

Xyce also supports `.OP` analysis statements. In **Xyce**, `.OP` should be considered as a shorthand for a single step DC sweep, in which all the default operating point values are used. One can also consider `.OP` analysis to be the operating point calculation which would occur as the intitial step to a transient calculation, without the subsequent time steps.

```
Diode Clipper Circuit
** Voltage Sources
VCC 1 0 5V VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940 D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
** GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+         IS = 4E-10
+         RS = .105
+          N = 1.48
+         TT = 8E-7
+        CJO = 1.95E-11
+         VJ = .4
+          M = .38
+         EG = 1.36
+        XTI = -8
+         KF = 0
+         AF = 1
+         FC = .9
+         BV = 600
+        IBV = 1E-4)
* .END
```

Figure 7.1. Diode clipper circuit netlist for DC sweep analysis.

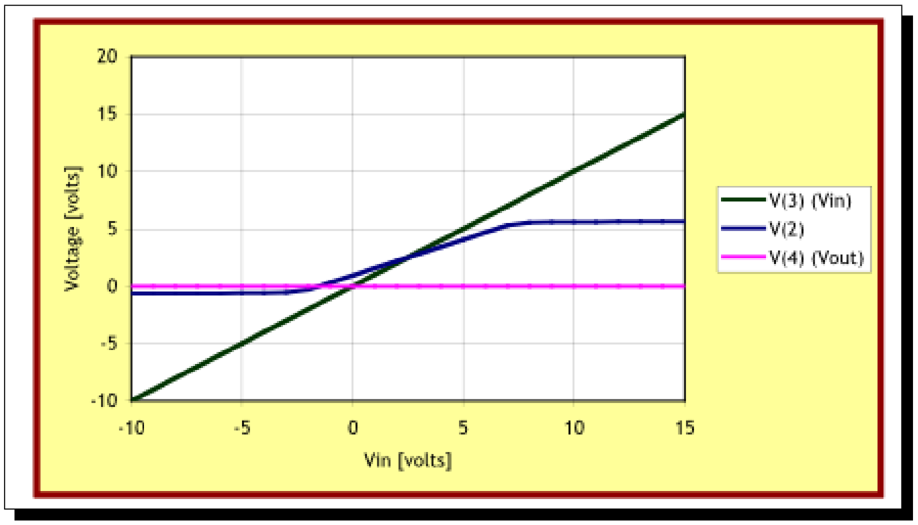


Figure 7.2. DC sweep voltages at Vin, node 2 and Vout.

This capability was mainly added so that the code would be able to handle legacy netlists which used this type of analysis statement. In most versions of SPICE, using `.OP` will result in extra output which is not available from a DC sweep. That additional output capability has not yet been implemented in **Xyce**.

7.3 Transient Analysis

The transient response analysis simulates the response of the circuit from `TIME=0` to a specified time. Throughout a transient analysis, any or all of the independent sources may have time-dependent values.

In **Xyce**, the transient analysis begins by performing its own bias point calculation at the beginning of the run, using the same method as used for DC sweep. This is required to set the initial conditions for the transient solution as the initial values of the sources may differ from the their DC values.

To run a transient simulation, the circuit netlist file must include a `.TRAN` command with the parameters required for the desired transient analysis (see the **Xyce** Reference Guide [2]). In addition, the netlist must contain one of the following:

- an independent, transient source (see Table 7.1),
- an initial condition on a reactive element, or
- a time-dependent behavioral modeling source (see Chapter 6)

Defining a Time-Dependent (transient) Source

Overview of Source Elements

Source elements, either voltage or current, are entered in the netlist file as described in the **Xyce** Reference Guide [2]. Table 7.1 list the time-dependent sources available in **Xyce** for either voltage or current. For voltage sources, the name is preceeded by the letter `V` while current sources are preceeded by the letter `I`.

Source Element Name	Description
EXP	Exponential Waveform
PULSE	Pulse Waveform
PWL	Piecewise Linear Waveform
SFFM	Frequency-modulated Waveform
SIN	Sinusoidal Waveform

Table 7.1. Summary of time-dependent sources supported by **Xyce**.

To use one of these time-dependent or transient sources, the user must place the source element line in the netlist and characterize the transient behavior using the appropriate parameters. Each transient source element has a separate set of parameters dependent on its transient behavior. In this way, the user can create analog sources which produce sine wave, square pulse, exponential pulse, single-frequency FM, and piecewise linear waveforms.

Defining Transient Sources

To define a transient source:

- Select one of the supported sources: independent voltage or current source.
- Choose a transient source type from Table 7.1.
- Provide the transient parameters (see the **Xyce** Reference Guide [2]) to fully define the source.

Below is an example of an independent sinusoidal voltage source in a circuit netlist. It creates a voltage source between nodes 1 and 5 that oscillates sinusoidally between -5V and +5V with a frequency of 50 KHz.

Example: `Vexample 1 5 SIN(-5V 5V 50KHz)`

Transient Calculation Time Steps

During the simulation, **Xyce** uses a calculation time step that is continuously adjusted for accuracy and efficiency (see [6]). During periods of circuit idleness the calculation time step is increased, and during dynamic portions of the waveform it is decreased. This release of **Xyce** does not allow the user to specify a maximum time step.

The internal calculation time steps used might not be consistent with the output time steps requested by the user. By default **Xyce** outputs solution results at every time step it calculates. If the user selects output timesteps via the `.OUTPUT` statement (see Chapter 9) then **Xyce** will output results for the closest time step that follows the time requested by the user. There is currently no mechanism for forcing **Xyce** to output at precise user-specified times.

Checkpointing and Restarting

The `.OPTIONS RESTART` command (in the netlist) is used to control all checkpoint output and restarting. Checkpointing and associated restart can be extremely useful for long simulations. In essence, **Xyce** allows the user to save the state of the simulation during a run (at intervals the user specifies) (*checkpointing*). This checkpoint data can then be read in to *restart* the simulation from any of the saved (*checkpointed*) time points.

Checkpointing Command Format

- `.OPTIONS RESTART PACK=<0|1> JOB=<job name> [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1>...]]]`

  `PACK=<0|1>` indicates whether the restart data files will contain byte packed data(1) or not(0). `JOB=<job name>` identifies the prefix for restart files. The actual restart files will be the job name appended with the current simulation time (e.g. `name1e-05` for `JOB=name` and simulation time 1e-05 seconds). Furthermore, the `INITIAL_INTERVAL=<interval>` identifies the initial interval time used for restart output. The `<tx ix>` intervals identify times (`tx`) at which the output interval (`ix`) will change. This functionality is identical that described for the `.OPTIONS OUTPUT` command (see Section 9.1).

- Example - generate checkpoints at every time step (default):

  ```
  .OPTIONS RESTART JOB=checkpt
  ```

- Example - generate checkpoints every 0.1 μs:

  ```
  .OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us
  ```

- Example - generate unpacked checkpoints every 0.1 μs:

  ```
  .OPTIONS RESTART PACK=0 JOB=checkpt INITIAL_INTERVAL=0.1us
  ```

- Example - Initial interval of 0.1 μs, at 1 μs in the simulation, change to interval of 0.5 μs, and at 10 μs change to an interval of 0.1 μs:

  ```
  .OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us 1us 0.5us
  + 10us 0.1us
  ```

Restarting Command Format

- `.OPTIONS RESTART <FILE=<filename> | JOB=<job name> START_TIME=<time>>`
  `+ [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]]`

To restart from an existing restart file, the file can be specified by using either the `FILE=<filename>` parameter to explicitly request a file or `JOB=<job name> START_TIME=<time>` to specify a file prefix and a specific time. The time must exactly match an output file time for the simulator to correctly load the file. To continue generating restart output files, `INITIAL_INTERVAL=<interval>` and following intervals can be appended to the command in the same format as described above.

- Example - Restart from checkpoint file at 0.133 μs:

```
.OPTIONS RESTART JOB=checkpt START_TIME=0.133us
```

- Example - Restart from checkpoint file at 0.133 μs :

```
.OPTIONS RESTART FILE=checkpt0.000000133
```

- Example - Restart from 0.133 μs and continue checkpointing at 0.1 μs intervals:

```
.OPTIONS RESTART FILE=checkpt0.000000133 JOB=checkpt_again
+ INITIAL_INTERVAL=0.1us
```

7.4 STEP Parametric Analysis

The `.STEP` command performs a parametric sweep for all the analyses of the circuit. When this command is invoked, all of the typical analysis, such as `.DC` or `.TRAN` analysis are performed at each parameter step.

This capability is very similar to the `STEP` capability in PSPICE and ChileSPICE, but not identical. Efforts will be made in future releases to make the `.STEP` capability in Xyce 100% compatible with those codes. In **Xyce**, `.STEP` can be used to sweep over any device instance or device model parameter, as well as the circuit temperature. Currently, there is not a capability for sweeping global parameters, as specified by a `.PARAM` statement.

Sweeping over a Device Instance Parameter

One specifies a `.STEP` analysis by simply adding a `.STEP` line to a netlist. `.STEP` by itself is not an adequate analysis specification, as it merely specifies an outer loop around the normal analysis. There needs to be a standard analysis line, such as `.TRAN` or `.DC` as well.

A typical `.STEP` line looks like this:

Example:
```
.STEP M1:L 7u 5u -1u
```

This has a very similar format to the `.DC` line. In this example, `M1:L` is the name of the parameter, `7u` is the initial value of the parameter, `5u` is the final value of the parameter, and `-1u` is the step size. Currently, **Xyce** can only handle linear sweeps.

The example uses `M1:L` as the parameter, but it could have been any model or instance parameter that existed in the circuit. Internally, **Xyce** handles the parameters for all device models and device instances in the same way. You can uniquely identify any parameter by specifying the device instance name, followed by a colon (:), followed by the specific parameter name. For example, all the MOSFET models have an instance parameter for the channel length, L. If you have a MOSFET instance specified in a netlist, named M1, then the full name for M1's channel length parameter is M1:L.

A simple application of `.STEP` to a device instance is given in figure 7.3. This is the same diode clipper circuit as was used in the transient analysis chapter, except that a single line (in red font) has been added. The `.STEP` line will cause **Xyce** to sweep the resistance of the resistor, R4, from 3.0 KOhms to 15.0 KOhms, in increments of 2.0 KOhms. This means that a total of seven transient simulations will be performed, each one with a different value for R4.

As the circuit is executed multiple times, the file output needs to be a little more sophisticated. The `.PRINT` statement is still used in much the same way as before. However, there is a separate *.prn output file for each `.STEP` increment.

```
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statement
.STEP R4:R 3.0K 15.0K 2.0K
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+         IS = 4E-10
+         RS = .105
+          N = 1.48
+         TT = 8E-7
+        CJO = 1.95E-11
+         VJ = .4
+          M = .38
+         EG = 1.36
+        XTI = -8
+         KF = 0
+         AF = 1
+         FC = .9
+         BV = 600
+        IBV = 1E-4)
*
.END
```

Figure 7.3. Diode clipper circuit netlist for step transient analysis.

The naming convention for `.STEP` simulation *.prn files is the same as in the non-`.STEP` case, except that the string "STEP*" is added to the name, where the "*" is an integer number indicating the step.

The example file given in figure 7.3 has a filename of clip.cir. The output files generated by the `.PRINT` statement are:

`clip.cir.STEP0.prn`	for `R4`	`= 3.0K`
`clip.cir.STEP1.prn`	for `R4`	`= 5.0K`
`clip.cir.STEP2.prn`	for `R4`	`= 7.0K`
`clip.cir.STEP3.prn`	for `R4`	`= 9.0K`
`clip.cir.STEP4.prn`	for `R4`	`= 11.0K`
`clip.cir.STEP5.prn`	for `R4`	`= 13.0K`
`clip.cir.STEP6.prn`	for `R4`	`= 15.0K`

These files will be similar in length, but not identical, as changing the resistance changes the numerical requirements a little bit, resulting in slightly different time step sizes.

Sweeping over a Device Model Parameter

Sweeping a model parameter can be done in an indentical manner to an instance parameter. Figure 7.4 contains the same circuit as in figure 7.3, but with a new `.STEP` line added. The new `.STEP` line refers to a model parameter, `D1N3940:IS`. Note that separate `.STEP` lines is the correct way to specify multiple parameters for `.STEP` analysis. Each parameter needs its own separate line. In this respect, the `.STEP` line syntax differs from the `.DC` line syntax.

Sweeping over Temperature

It is also possible to sweep over temperature. To do so, simply specify `temp` as the parameter name. It will work in the same manner as `.STEP` when applied to model and instance parameters.

Special cases: Sweeping Independent Sources, Resistors, Capacitors

For some devices, there is generally only one parameter that one would want to actually sweep. For example, a linear resistor's only parameter of interest is the resistance, R. Similarly, for a DC voltage or current source, one is usually only interested in the magnitude of the source. Finally, linear capacitors generally only have the capacitance, C, as a parameter of interest. To make things easier for the user, these three types of devices have default parameters. Examples of usage are given below.

```
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statements
.STEP R4:R 3.0K 15.0K 2.0K
.STEP D1N3940:IS 2.0e-10 6.0e-10 2.0e-10
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+         IS = 4E-10
+         RS = .105
+          N = 1.48
+         TT = 8E-7
+        CJO = 1.95E-11
+         VJ = .4
+          M = .38
+         EG = 1.36
+        XTI = -8
+         KF = 0
+         AF = 1
+         FC = .9
+         BV = 600
+        IBV = 1E-4)
*
.END
```

Figure 7.4. Diode clipper circuit netlist for 2-step transient analysis.

Example:
```
.STEP R4   3.0K   15.0K   2.0K
.STEP VCC  4.0     6.0    1.0
.STEP ICC  4.0     6.0    1.0
.STEP C1   0.45u   0.50u  0.1u
```

Independent sources require some extra explanation. There are a number of different types of independent source, and only some of them have default parameters. Sources which are subject to `.DC` sweeps (swept sources) do not have a default parameter, as this could easily lead to infinite loops. The various independent source defaults are defined in the table.

Source Type	**Default**
Sinusoidal source	V0 (DC value, Offset)
Exponential source	V1 (DC value, Initial value)
Pulsed source	V2 (Pulsed value)
Constant, or DC source	V0 (Constant value)
Piecewise Linear source	No default
SFFM source	No default
Swept source (specified on a `.DC` line)	No default

Table 7.2: Default parameters for independent sources.

This page is left intentionally blank

8. Using Homotopy Algorithms to Obtain Operating Points

Chapter Overview

This chapter includes the following sections:

- Section 8.1, *Homotopy Algorithms Overview*
- Section 8.2, *Examples*

8.1 Homotopy Algorithms Overview

The most difficult type of numerical nonlinear circuit problem to solve is a DC operating point. Unlike transient analysis, DC operating point analysis cannot rely on the results of a previous time step. Also, operating points often have multiple solutions, both valid and invalid.

Homotopy methods can often provide solutions to difficult nonlinear problems when other, more conventional numerical methods fail. In recent years, these techniques have been applied to circuit analysis. As of the **Xyce** Version 2.0 release, some of these algorithms have been added to **Xyce**. This chapter gives an introduction to the usage of homotopy algorithms (also called continuation algorithms) in **Xyce**. For a more complete description of solver options, see the Xyce Reference Guide [2].

HOMOTOPY Algorithms Available in **Xyce**

There are two general types of homotopy which are available in **Xyce**. The first (which is set with `.options nonlin continuation=1`), is a simple natural parameter homotopy, in which the homotopy parameter is an already-defined input parameter to a device model or instance. This algorithm can be useful, but often is not. The most obvious natural parameters to use (the magnitudes of independent sources) tend to lead to turning points in the continuation.

The second is an algorithm which is designed especially for MOSFET circuits [7]. This algorithm involves two internal MOSFET model parameters, one for the MOSFET gain, and the other for the nonlinearity of the current-voltage relationship. This algorithm is invoked with `.options nonlin continuation=2`. This algorithm has proved to be very effective in large MOSFET circuits.

8.2 Examples

MOSFET Homotopy

Figure 8.1 contains a MOSFET homotopy example netlist. Note that this is a usage example - the circuit itself does not require homotopy to run. Circuits which are complex enough to require homotopy would not fit on a single page. The lines pertainent to the homotopy algorithm are highlighed in red.

Explanation of Parameters, Best Practice

Note that this example shows one set of options, but there are a number of other combinations of options that will work.

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3
.options linsol ksparse=1


* HOMOTOPY Options
.options device voltlim=0

.options nonlin continuation=2

.options loca stepper=0 predictor=0 stepcontrol=1
+ initialvalue=0.0 minvalue=-1.0 maxvalue=1.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8.1. Example MOSFET homotopy netlist.

There are a number of ”best practice” rules, which are illustrated by the example in figure 8.1. They are:

- `voltlim=0`. This is generally required - the homotopy algorithms will usually break if this is not set.
- `continuation=2`. This specifies that we are using the special MOSFET homotopy. This is a 2-pass homotopy, in which first a parameter having to do with the gain is swept from 0 to 1, and then a parameter relating to the nonlinearity of the transfer curve is swept from 0 to 1.
- `initialvalue=0.0`. This is required.
- `maxvalue=1.0`. This is required.
- `stepcontrol=1`. This specifies that the homotopy steps are adaptive, rather than constant. This is recommended.
- `maxsteps=100`. This sets the maximum number of continuation steps for each parameter. For the special MOSFET continuation (which has 2 parameters), this means a maximum of 200 steps.
- `maxnliters=200`. This is the maximum number of nonlinear iterations, and has precedence over the similar number which can be set on the .options nonlin line.
- `aggressiveness=1.0`. This refers to the step size control algorithm, and the value of this parameter can be anything from 0.0 to 1.0. 1.0 is the most aggressive. In practice, try starting with this set to 1.0. If the solver fails, then reset to a smaller number.

Natural Parameter Homotopy

Figure 8.2 contains a natural parameter homotopy netlist. It is the same circuit as was used in figure 8.1, except that some of the parameters are different. As before, the lines pertainent to the homotopy algorithm are highlighed in red.

Explanation of Parameters, Best Practice

There are a few differences between the netlist in figure 8.1 and figure 8.2. They are:

- `continuation=1`. Sets the algorithm to use natural parameter homotopy.
- `conparam=VDDdev`. If using natural parameter homotopy, this is required. It sets which input parameter to use. The parameter name is subject to the same rules as parameter used by the `.STEP` capability. (See section 7.4). In this case the parameter is the magnitude of the DC voltage source, VDDdev. For this type of voltage source, it was possible to use the default device parameter (see section 7.4)

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3
.options linsol ksparse=1

* HOMOTOPY Options
.options device voltlim=0

.options nonlin continuation=1

.options loca stepper=0 predictor=0 stepcontrol=1
+ conparam=VDDdev
+ initialvalue=0.0 minvalue=-1.0 maxvalue=5.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8.2. Example natural parameter homotopy netlist.

Using the magnitudes of independent voltage and current sources is a fairly obvious approach. Unfortunately, it doesn't seem to work very well in practice.

9. Results Output and Evaluation Options

Chapter Overview

This chapter illustrates how to output simulation results to data or output files.

- Section 9.1, *Control of Results Output*
- Section 9.2, *Additional Output Options*
- Section 9.3, *Evaluating Solution Results*

9.1 Control of Results Output

Xyce supports only one solution output command, `.PRINT`. `.PRINT` is quite flexible, and supports several output formats.

`.PRINT` Command

The `.PRINT` command sends the analysis results to an output file. **Xyce** supports several options on the `.PRINT` line of netlists that control the format of the output. The syntax for the command is as follows:

- `.PRINT <analysis type> [options] <output variable(s)>`

Example: `.PRINT TRAN FILE=Output.prn V(3) V(2) V(4)`

Table 9.1 gives the various options currently available to the `.PRINT` command. For further information, see the **Xyce** Reference Guide [2].

9.2 Additional Output Options

`.OPTIONS OUTPUT` Command

The main purpose of the `.OPTIONS OUTPUT` command is to provide control of the frequency at which data is written to files specified by `.PRINT TRAN` commands. This can be especially useful in controlling the size of the results file for simulations which required a large number of time steps. An additional benefit is that reducing the output frequency from the default, which outputs results at every time-step, can improve performance. The format for controlling the output frequency is:

- `.OPTIONS OUTPUT INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]`

where `INITIAL_INTERVAL=<interval>` specifies the starting interval time for output and `<tx ix>` specifies later simulation times (`tx`) where the output interval will change to (`ix`).

The following example shows the output being requested (via the netlist `.OPTIONS OUTPUT` command) every .1μs for the first 10μs, every 1μs for the next 10μs, and every 5μs for the remainder of the simulation:

Example: `.OPTIONS OUTPUT INITIAL_INTERVAL=.1us 10us 1us 20us 5us`

Option...	Action...
`FORMAT=<STD|NOINDEX|PROBE>`	Controls the output format. The `STD` format outputs data in standard columns. The `NOINDEX` format is the same as the standard format except that the index column is omitted. The `PROBE` format specifies that the output should be formatted to be compatible with the PSpice Probe plotting utility. The *default* is `STD`.
`FILE=<output filename>`	Allows the user to specify the output filename. The *default* is the netlist filename with the characters "`.prn`" appended (e.g., `foo.cir.prn` where `foo.cir` was the input netlist filename.
`WIDTH=<print field width>`	Allows the user to control the column width for the output data.
`PRECISION=<floating point precision>`	Controls the number of significant digits past the decimal point.
`FILTER=<filter floor value>`	Specifies the absolute value below which output variables will be printed as 0.0.
`DELIMITER=<TAB|COMMA>`	Specifies an alternate delimiter between columns of output in the STD output format.

Table 9.1. .PRINT command options.

Note: **Xyce** will output data at the next time that is greater-than or equal to the current interval time. This means that output might not correspond exactly to the time intervals due to the adaptive time stepping algorithm.

9.3 Evaluating Solution Results

This section describes how to view graphical waveform analysis of the simulation results generated by **Xyce**. You can use the solution output features of **Xyce** in conjunction with graphing tools (e.g., TecPlot, gnuplot, MS Excel, etc.) to analyze graphically the waveform data created by a **Xyce** circuit simulation (see Figure 9.1 below for an example plot using TecPlot, `http://www.amtec.com`). In addition, by using the `FORMAT=PROBE` option to the `.PRINT` command, **Xyce** is able to output `.csd` files which can be read by the PSpice Probe utility to view the results. See the PSpice Users Guide [3] for instructions on using the Probe tool, and the **Xyce** Reference Guide [2] for details on the options to the `.PRINT` command.

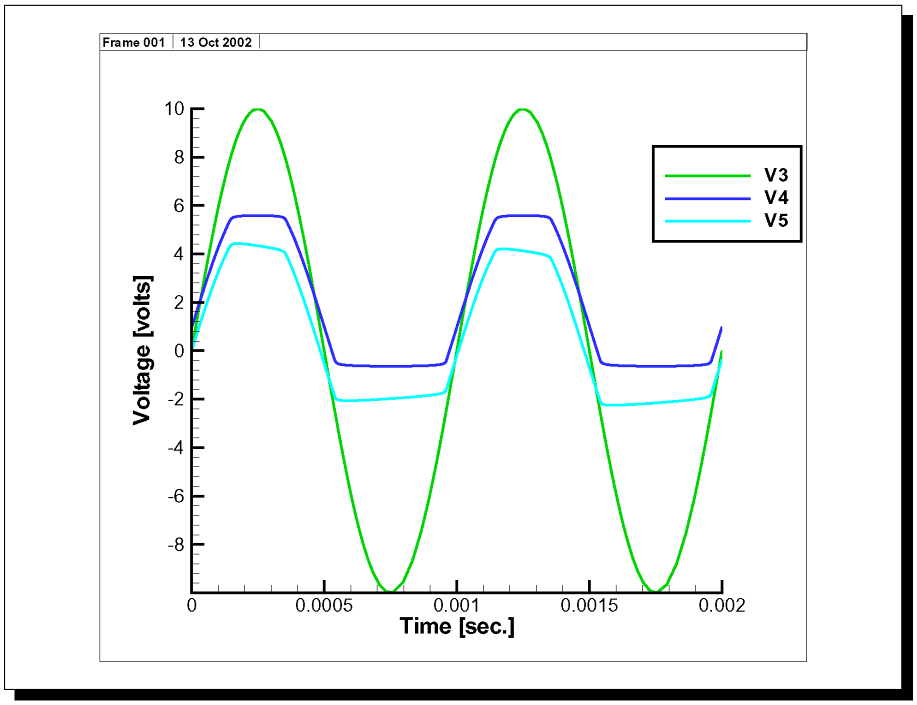


Figure 9.1. TecPlot plot of diode clipper circuit transient response from **Xyce** `.prn` file.

Xyce produces two types of output: the simulation output file and the waveform data file. The calculations and results reported in the simulation output file can be thought of as an audit trail of the simulation. However, graphical analysis of data in the waveform data file is the most useful and accommodating way to evaluate simulation results.

10. Guidance for Running **Xyce** in Parallel

Chapter Overview

This chapter gives guidance on how to run a parallel version of **Xyce**. It includes the following sections:

- Section 10.1, *Introduction*
- Section 10.2, *Mechanics*
- Section 10.3, *Problem Size*
- Section 10.4, *Linear Solver Options*
- Section 10.5, *Partitioning Options*

10.1 Introduction

Xyce has been designed, from the ground up, to support a message-passing parallel implementation. As such, **Xyce** is unique among circuit simulation tools, and many of the issues pertainent to running in parallel are still research issues. However, **Xyce** is now mature enough that some general principles have emerged, for effectively running problems in a parallel environment.

10.2 Mechanics

Parallel simulations must be run from the command line. Details of how to do this are given in section 2.2.

10.3 Problem Size

Due to the overhead of interprocessor communication, running **Xyce** in parallel is only useful for large circuit problems. Also, for any problem size, there is an optimal number of processors. As one increases the processor count, the amount of communication required increases and the work per processor decreases. This increase in communication will slow a simulation down, while the reduction in work per processor will have the reverse effect. If the number of processors is too large, the benefit of distributing the problem will be outweighed by the high cost of communication overhead. Such a limit exists for every size of problem, and increasing the processor count beyond this point is counterproductive. This issue is most pronounced for platforms with a high communication cost, such as Beowulf clusters.

Smallest Possible Problem Size

Circuits are comprised of a discrete set of components (voltage nodes, devices, etc.). To run in parallel, it is preferable that **Xyce** be able to put at least one discrete part of the problem on each processor. In practice, this means that the number of processors should be less than the number of nodes in the circuit. **Xyce** is capable of simulating smaller problems, but it is not recommended, due to solver instabilities.

Ideal Problem Size

To take full advantage of **Xyce**'s parallel capability, the problem should be relatively large. A good metric for estimating how many processors one should use is the number of devices per processor. The ideal number of devices per processor is very machine and problem dependent. For machines such as SGI Challenger platforms, with relatively fast communication speeds in comparison to processor performance, reasonable speedups can be seen for 100's of devices per processor. For Beowulf clusters with relatively slow communication speeds in comparison to processor performance, 1000's of devices per processor are required to achieve reasonable speedups in parallel. These numbers are very problem-dependent, as the effectiveness of the load balance and partitioning can vary a lot.

10.4 Linear Solver Options

There are several different linear solvers available in **Xyce** version 2.0. They are:

- The Aztec iterative solver library.
- SuperLU.
- Kudert Sparse (KSparse).

For **Xyce** version 2.0, only Aztec is available in parallel. For future versions of **Xyce**, a parallel direct solver will be available for moderate problem sizes.

10.5 Partitioning Options

Xyce currently has two graph partitioning options available. These partitioning utilities subdivide the circuit problem into sections that are then distributed to the processors. A good partition can have a dramatic effect on the parallel performance of circuit simulation. There are two key components to a good partition:

- Effective load balance.
- Minimizing communication overhead.

An effective load balance ensures that the computational load of the calculation is equally distributed among the available processors. Minimizing communication overhead seeks to distribute the problem in a way that reduces the impact of underlying message passing during the simulation run. For runs with a small number of devices per processor the communication overhead becomes the critical issue, while for runs with larger numbers of devices per processor the load balancing becomes more important. **Xyce** has integrated within it two partitioning libraries - the **Chaco** static partitioner and the **ZOLTAN** library of parallel partitioning heuristics.

Chaco Static Partitioning of Circuit

Chaco is accessible using the `'.OPTIONS PARALLEL PARTITIONER=0'` line in the netlist. By adding this line to the netlist, **Chaco** will be used to partition the initial circuit before it is distributed to processors. **Chaco** partitioning can be controlled through a `'Chaco_User_Params'` file that must be in the execution directory. See the **Chaco User Guide** [8] for details.

Currently, one parameter is available for **Chaco** partitioning: using `'DISTRIBINDSRCNODES=0'` as a parallel option can be very effective for an improved partitioning, especially for large digital circuits. Adding this parameter to the parallel options allows **Xyce** to replicate independent voltage sources across all processors that have devices connected to them. In many cases this can dramatically reduce the interprocessor connectivity and reduce the communication overhead substantially. The parameter value determines the fractional degree of connectivity for a voltage source required before allowing it to be replicated. Therefore setting to zero allows all voltage sources to be replicated as necessary. Setting the value to larger numbers has proven to be ineffective.

Due to the renaming and distribution of some independent sources and their associated voltage nodes, restarting can only be done when the restarted run uses an identical number of processors and partitioning as the job that created the restart files.

Zoltan Partitioning of the Linear System

Zoltan is accessible through the `'.OPTIONS LINSOL'` control line in the netlist. By adding an options `'TR_LOADBALANCE=1'` to the linear options, the linear system is statically load balanced based on the graph of the Jacobian matrix. The local system is also reordered based on nested dissection which should improve conditioning and minimize fill. These techniques can be very effective for improving the efficiency of the iterative linear solvers. See the **Zoltan User Guide** [9] for details.

Singleton Filtering of the Linear System

Singleton filtering refers to the reduction of the linear system through removal of all rows and columns with single non-zero entries. The values associated with these removed entries can be resolved as pre/post solve linear operations. A by-product of this reduction is a more tractable, sparser linear system for both the load balancing and linear solver algorithms. This functionality is turned on by adding the `'TR_SINGLETON_FILTER=1'` option to the `'.OPTIONS LINSOL'` control line in the netlist. It is also recommended to add the `'TR_SOLVERMAP=1 TR_REINDEX=1 USE_IFPACK_PRECOND=1'` to the `'.OPTIONS LINSOL'` as well to improve the solver robustness with singleton filtering. In the future, it is expected these options will be use as default.

10.6 Recommended Partitioning and Solver Options

A recommended set of options for parallel problems is given below. These options include singleton filtering as well as Chaco and Zoltan partitioning of the circuit and linear system respectively. The additional settings can improve the performance of the preconditioned linear solver.

- `.OPTIONS PARALLEL PARTITIONER=0 DISTRIBINDSRCNODES=0`
- `.OPTIONS LINSOL TR_LOADBALANCE=1 TR_SINGLETON_FILTER=1 TR_SOLVERMAP=1 TR_REINDEX=1 USE_IFPACK_PRECOND=1`

An important caveat is that the performance and convergence of the linear solver for parallel problems can be substantially less robust for some circuits. This is a known issue with parallel iterative solution of linear problems. If **Xyce** is not performing as expected for these problems, please consult the developer team.

This page is left intentionally blank

11. PDE Device Simulation with **Xyce**

Chapter Overview

This chapter gives guidance on how to use the mesh-based device simulation capability of **Xyce**. It includes the following sections:

- Section 11.1, *Introduction*
- Section 11.2, *One Dimensional Example*
- Section 11.3, *Two Dimensional Example*
- Section 11.4, *Doping Profile*
- Section 11.5, *Electrodes*
- Section 11.6, *Meshing*
- Section 11.7, *Mobility Models*
- Section 11.8, *Bulk Materials*
- Section 11.9, *Solver Options*
- Section 11.10, *Output and Visualization*

11.1 Introduction

This chapter describes how to use the mesh-based device simulation funcionality of **Xyce**. This capability is based on the solution a coupled set of partial differential equations (PDEs), discretized on a mesh such as the one in Figure 11.1. While the rest of **Xyce** is intended to be similar to analog circuit simulators such as SPICE, the PDE device capability is intended to be similar to well-known device simulators such as PISCES [10] and DaVinci [11].

Two different PDE devices are available **Xyce**: a one-dimensional device and a two-dimensional device. These two devices have been implemented in a manner which allows them to be invoked in the same way as a conventional lumped parameter circuit device. Generally, this capability is intended for very detailed simulation of semiconductor devices, such as diodes, bipolar transistors, and MOSFETs. One possible application of this capability is the evaluation and/or analysis of conventional SPICE-style lumped parameter models.

NOTE: As of **Xyce** Release 2.0, the PDE-based devices should still be considered to be a beta-level capability. The primary focus of **Xyce** has been traditional analog circuit simulation, so the PDE devices have not been subject to the same level of testing as the traditional, SPICE-style devices. The PDE device simulator in **Xyce** should be regarded as a prototype for Charon, a high performance 3D device simulator that is under development at Sandia.

NOTE: To avoid conflicts, some of the netlist specification (particularly the use of brackets [] in the doping and electrode specifications) will be changed for **Xyce** Release 2.1.

Equations

The equations of device simulation are described by many references including Kramer [12] and Selberherr [13]. The most common formulation, and the one that is used in **Xyce**, is the drift-diffusion (DD) formulation. This formulation consists of three coupled PDE's: a single Poisson equation for electrostatic potential and two continuity equations; one each for electrons and holes.

Poisson equation

The electrostatic potential ϕ satisfies Poisson's equation:

$$-\nabla \cdot (\epsilon \nabla \phi(x)) = \rho(x) \tag{11.1}$$

where ρ is the charge density and ϵ is the permittivity of the material. For semiconductor devices, the charge density is determined by the local carrier densities and the local doping,

$$\rho(x) = q(p(x) - n(x) + C(x)) \tag{11.2}$$

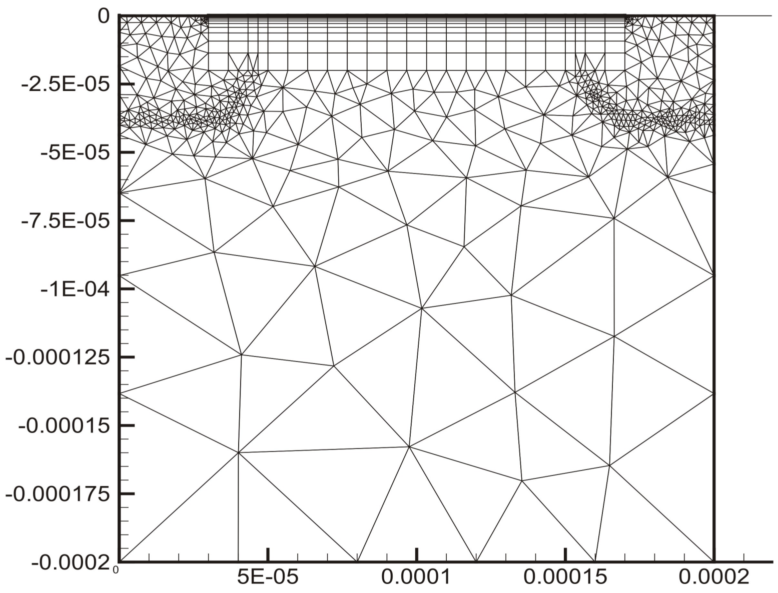


Figure 11.1. MOSFET Mesh Example.

Here $p(x)$ is the spatially-dependent concentration of holes, $n(x)$ the concentration of electrons, and q the magnitude of the charge on an electron. $C(x)$ is the total doping concentration, which can also be represented as $C(x) = N_D^+(x) - N_A^-(x)$, where N_D^+ the concentration of positively ionized donors, N_A^- the concentration of negatively ionized acceptors.

Species continuity equations

The continuity equations relate the convective derivative of the species concentrations to the creation and destruction of particles ("recombination/generation").

$$\frac{\partial n(x)}{\partial t} + \nabla \cdot \Gamma_n = -R(x) \tag{11.3}$$

$$\frac{\partial p(x)}{\partial t} + \nabla \cdot \Gamma_p = -R(x) \tag{11.4}$$

Here n is the electron concentration and p is the hole concentration. R is the recombination rate for both species. Γ_n and Γ_p are particle fluxes for electrons and holes, respectively. The sign of R is chosen because R is usually expressed as a recombination rate, and is positive if particles are annihilating. The right hand sides are equal since creation and destruction of carriers occurs in pairs.

One way in which the drift-diffusion model differs from other common formulations is the manner in which the quantities Γ_n and Γ_p are determined. The expressions used are:

$$\Gamma_n = n(x)\mu_n E(x) + D_n \nabla n(x) \tag{11.5}$$

$$\Gamma_p = p(x)\mu_p E(x) + D_p \nabla p(x) \tag{11.6}$$

Here μ_n, μ_p are mobilities for electrons and holes, and D_n, D_p are diffusion constants. $E(x)$ is the electric field, which is given by the gradient of the potential, or $-\partial\phi/\partial x$.

Discretization

Xyce uses a box-integration discretization, with the Scharfetter-Gummel method for modeling the flux of charged species. This method has been described in detail elsewhere [12] [13] [14], so it will not be described here.

11.2 One Dimensional Example

The one-dimensional device was the first PDE-based device to be implemented in **Xyce**. The single dimension limits its usefulness, but its simplicity makes it a good device to use for a preliminary example. One dimensional devices are almost always two-terminal diodes, and this fact allows for assumptions which simplify the specification and shorten the parameter list of the device.

An example netlist, for a PDE simulation of a one-dimensional diode, is shown in Figure 11.2. The corresponding schematic is in Figure 11.3. The circuit is a regulator circuit, and is based on the principle that connecting one or more diodes in series with a resistor and a power supply will produce a relatively constant voltage. The input voltage (node 2) is a sinewave, with a frequency of 50 Hz and an amplitude of 1 V. The expected output (node 3) should be a (mostly) flat signal.

```
PDE Diode Regulator Circuit
VP 1 0 PULSE(0 5 0.0 2.0e-2 0.0 1.0e+20 1.2e+20)
VF 2 1 SIN(0 1 50 2.0e-2)
VT1 4 0 0V
R1 2 3 1k

 * PDE Device
Z1 3 4 DIODE na=1.0e19 nd=1.0e19 graded=0
+ l=5.0e-4 nx=101

 .MODEL DIODE  ZOD

.TRAN 1.0e-3 12.0e-2
.print TRAN format=tecplot
+ v(1) v(2) v(3) v(4) I(VF) I(VT1)

.options NONLIN maxstep=100 maxsearchstep=3
+ searchmethod=2  nox=0

.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
+ resettrannls=0

.END
```

Figure 11.2. One dimensional diode netlist. This circuit is a voltage regulator. The input signal should be a sinewave, while the output signal should be nearly flat. For the result of this netlist, see Figure 11.4

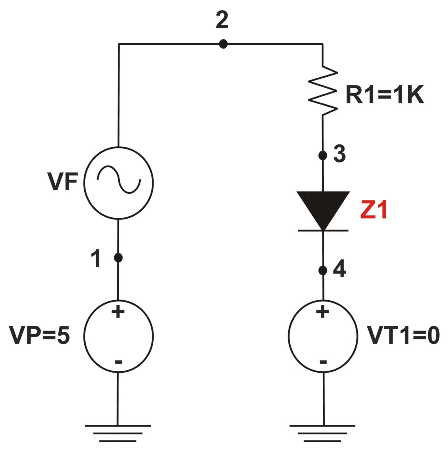


Figure 11.3. Voltage regulator schematic. The diode, Z1, is the PDE device in this example.

Netlist Explanation

In Figure 11.2, the PDE device instance line is in red, while the PDE device model line is in blue. Currently, there are almost no model parameters for PDE devices. The model line serves only to set the level. The default level is 1, for a one-dimensional PDE device. Two dimensional devices are invoked by setting `level=2`. Note that in this example, the level is not explicitly set, and so the default (1) is used.

The instance line is where most of the specific parameters are set for a PDE device. In this example, the line appears as:

```
Z1 3 4 DIODE na=1.0e19 nd=1.0e19 graded=0 l=5.0e-4 nx=101
```

`na` and `nd` are doping parameters, and represent the majority carrier doping levels on the N-side and the P-side of the junction, respectively. `graded=0` is also a doping parameter, and specifies that the junction is not a graded junction, but is an abrupt step-function junction instead. `l=5.0e-4` specifies the length of the device, in cm. `nx=101` specifes that there are 101 mesh points, including the two endpoints. For the one-dimensional PDE device, the mesh is always uniform, so the size of each mesh cell, Δx will be:

$$\Delta x = \frac{l}{nx-1} = \frac{\text{5.0e-4 cm}}{100} = \text{5.0e-6 cm} \tag{11.7}$$

The mesh points $i = 0 - 101$ will have the following locations, x_i:

$$\begin{aligned} x_i &= i\Delta x \\ x_o &= \text{0.0 cm} \\ x_1 &= \text{5.0e-6 cm} \\ &\cdot \end{aligned}$$

$$\vdots$$
$$x_{101} = \text{5.0e-4 cm}$$

Boundary Conditions and Doping Profile

Note that nothing has been specified in the example netlist about electrodes, or boundary conditions, and that the doping specification is minimal. This is because the example relies a lot on default parameters. A one dimensional PDE device can only have exactly 2 electrodes connected to the circuit. These two electrodes are at opposite ends of the domain, one at the first mesh point (`x=0.0 cm`, `i=0`) and the other at the opposite end of the domain, at the last mesh point (`x=5.0e-4 cm`, `i=101`).

The electrode associated with the first mesh point (`x=0.0 cm`) is connected to the *second* circuit node on the instance line, while the electrode associated with the last mesh point (`x=l`) is connected to the *first* circuit node on the instance line. For the doping used in this example, the junction is in the exact center of the device (`x=l/2`), and the n-side is the region defined by `x<l/2`, and the p-side is the region defined by `x>l/2`. This default doping, along with the electrode-circuit connectivity, result in the one-dimensional PDE device to behave like a traditional SPICE-style diode. For a complete discussion of how to specify a doping profile see section 11.4. For a complete discussion of how to specify electrodes in detail (including boundary conditions), see section 11.5.

Results

The transient result of this circuit is shown in Figure 11.4. The input signal (node 2) is represented by the blue line, and the output signal (node 3) is represented by the red line. The voltage drop across the diode is nearly the same for a wide range of currents, and is approximately 0.67 V. The voltage drop across the series resistor, `R1`, is much more sensitive to the current magnitude, and so most of the voltage variation of the input sinewave is accounted for by `R1`.

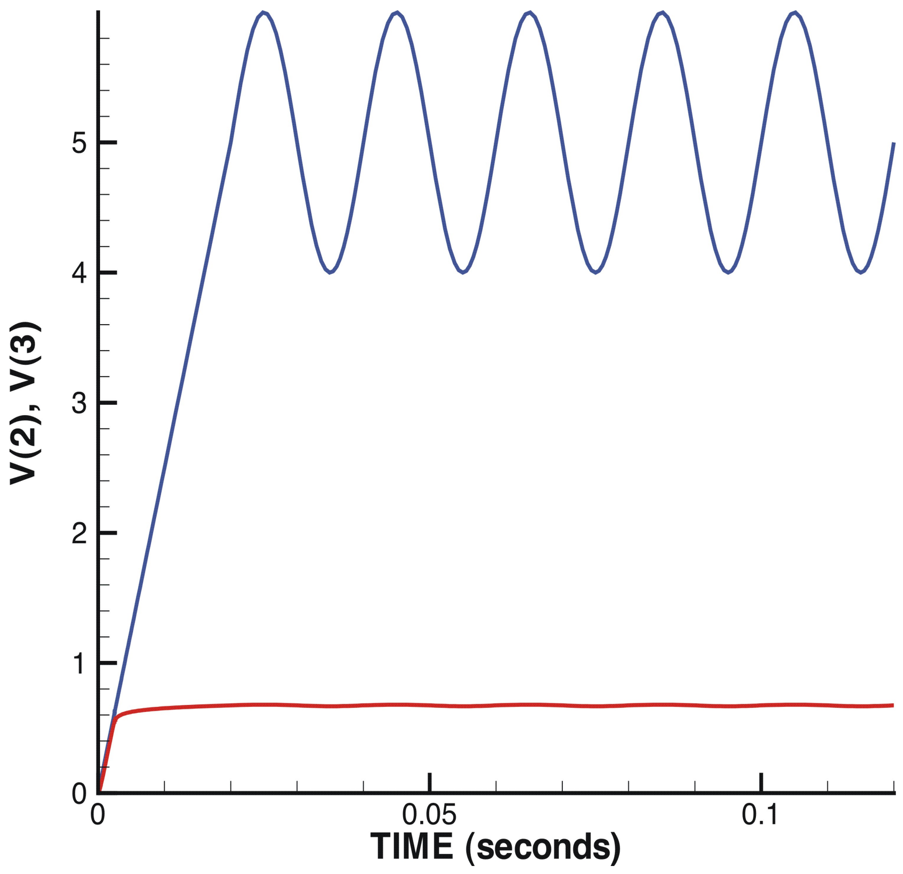


Figure 11.4. Transient result for the voltage regulator circuit in Figure 11.2. The input voltage is represented by the blue line, while the output voltage is given by the red line.

11.3 Two Dimensional Example

An example netlist, for a PDE simulation of a two-dimensional bipolar transiistor, is shown in Figure 11.5. As before, the PDE device instance line is in red, while the PDE device model line is in blue. In this case, note that the level has been specified on the model line, and it has been set to 2. This is required for the two-dimensional device. This particular example is a DC sweep of a bipolar transistor device. A schematic, illustrating this circuit is shown in Figure 11.6.

Netlist Explanation

The two-dimensional device can have 2-4 electrodes. (this limitation will be relaxed in future versions of **Xyce**) In this example there are three; node 5, node 3 and node 7. These correspond to the three names on the ”node” line, which appears as:

```
+ node = [name = collector, base, emitter]
```

This line specifies that node 5 is connected to an electrode named ”collector”, node 3 is connected to an electrode named ”base”, and node 7 is connected to an electrode named ”emitter”. Although this example only contains the electrode names, the ”node” specifcation can contains a lot of information. For a full explanation of all the electrode parameters, see section 11.5.

The next line contains parameters concerned with plotting the results, and appears as follows:

```
+ tecplotlevel=2 txtdatalevel=1
```

Note that these are not related to the output specified by `.PRINT`, which outputs circuit data. The `tecplotlevel` command enables files to be output which are readable by tecplot. Tecplot can then be used to create contour plots of quantities such as the electron density, electrostatic potential and the doping profile. Figures 11.7 and 11.8 contain examples of tecplot-generated contour plots, which were generated from the results of this example.

The `txtdatalevel` command enables a text file with volume averaged information to be output to a file. Currently, both of these output files will be updated at each time step or DC sweep step.

The next line, `mobmodel=arora`, specifies which mobility model to use. For more detail on available mobility models, see section 11.7.

The last two lines, specify the mesh of the device, and are given by:

```
+ l=2.0e-3 w=1.0e-3
+ nx=30 ny=15
```

```
Two Dimensional Example
VPOS   1 0 DC 5V
VBB    6 0 DC -2V
RE     1 2 2K
RB     3 4 190K

 Z1BJT 5 3 7 PDEBJT meshfile=internal.msh
+ node = [name = collector, base, emitter]
+ tecplotlevel=2 txtdatalevel=1
+ mobmodel=arora
+ l=2.0e-3  w=1.0e-3
+ nx=30     ny=15

* Zero volt sources acting as an ammeter to measure the
* base, collector, and emmitter currents, respectively
VMON1 4 6 0
VMON2 5 0 0
VMON3 2 7 0

.MODEL PDEBJT   ZOD  level=2

.DC VPOS 0.0 12.0 0.5 VBB -2.0 -2.0 1.0

.options LINSOL ksparse=1
.options NONLIN maxstep=70 maxsearchstep=1
+ searchmethod=2 in_forcing=0 nlstrategy=0
+ directlinsolv=1

.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
+ firstdcopstep=0 lastdcopstep=1

.PRINT DC V(1) I(VMON1) I(VMON2) I(VMON3)

.END
```

Figure 11.5. Two-dimensional BJT netlist. Some results of this netlist can be found in Figures 11.7 and 11.8.

This numbers are used in nearly the same way as the `l` and `nx` parameters were used in the one-dimensional case. The mesh is cartesian, and the spacing is uniform.

Doping Profile

As in the one-dimensional example, the two-dimensional example in figure 11.5 does not specify anything about the doping profile, and thus relies upon defaults. In this case there are three specified electrodes, which by default results in the doping profile of the bipolar junction transistor (BJT). For a complete description of how to specify a doping profile in detail, see section 11.4. This section also describes the various default impurity profiles.

Boundary Conditions and Electrode Configuration

As in the one-dimensional example, the two-dimensional example in figure 11.5 does not specify anything about the electrode configuration or the boundary conditions, and relies on default settings. To be consistent with the default 3-terminal doping, the device has terminals that correspond to that of a BJT. All three electrodes (collector, base, emitter) are along the top of the device.

By default all electrodes are considered to be neutral contacts. The boundary conditions applied to the electron density, hole density and electrostatic potential are all Dirichlet conditions.

For a complete discussion of how to specify electrodes in detail (including boundary conditions), see section 11.5.

Results

Results for the two-dimensional example can be found in Figures 11.7, 11.8 and 11.9. The first two figures are contour plots of the electrostatic potential. The first one corresponds to the first DC sweep step, where VPOS is set to 0.0 Volts. The second one corresponds to the final DC sweep step, in which VPOS has a value of 12.0 volts. The voltage source VPOS applies a voltage to the emitter load resistor, RE, so some of the 12.0V is dropped accross RE, an the rest is applied to the BJT.

The third figure is an I-V curve of the dependence of the three terminal currents on applied emitter voltage. For the entire sweep, a negative voltage of 2.0 V has been applied to the base load resistor, and as this transistor is a PNP transistor, this results in the transistor being in an “on” state. The emitter-collector current varries nearly linearly with the applied emitter voltage. Also, the three currents sum to nearly zero, which one would expect because of current conservation.

Note that the mesh is visible in Figure 11.7, and was generated using the internal “uniform mesh” option. Generally using this sort of mesh will work numerically, in that **Xyce** will converge to an answer. However, this mesh will probably not produce a very accurate result, as it does not resolve the depletion regions very well. In order to obtain better accuracy, either a finer uniform mesh would need to be used, or a nonuniform mesh, refined in around the depletion regions should be used. As described in section 11.6, refined, nonuniform meshes must be read in from an external mesh generator, such as the SGFramework [12].

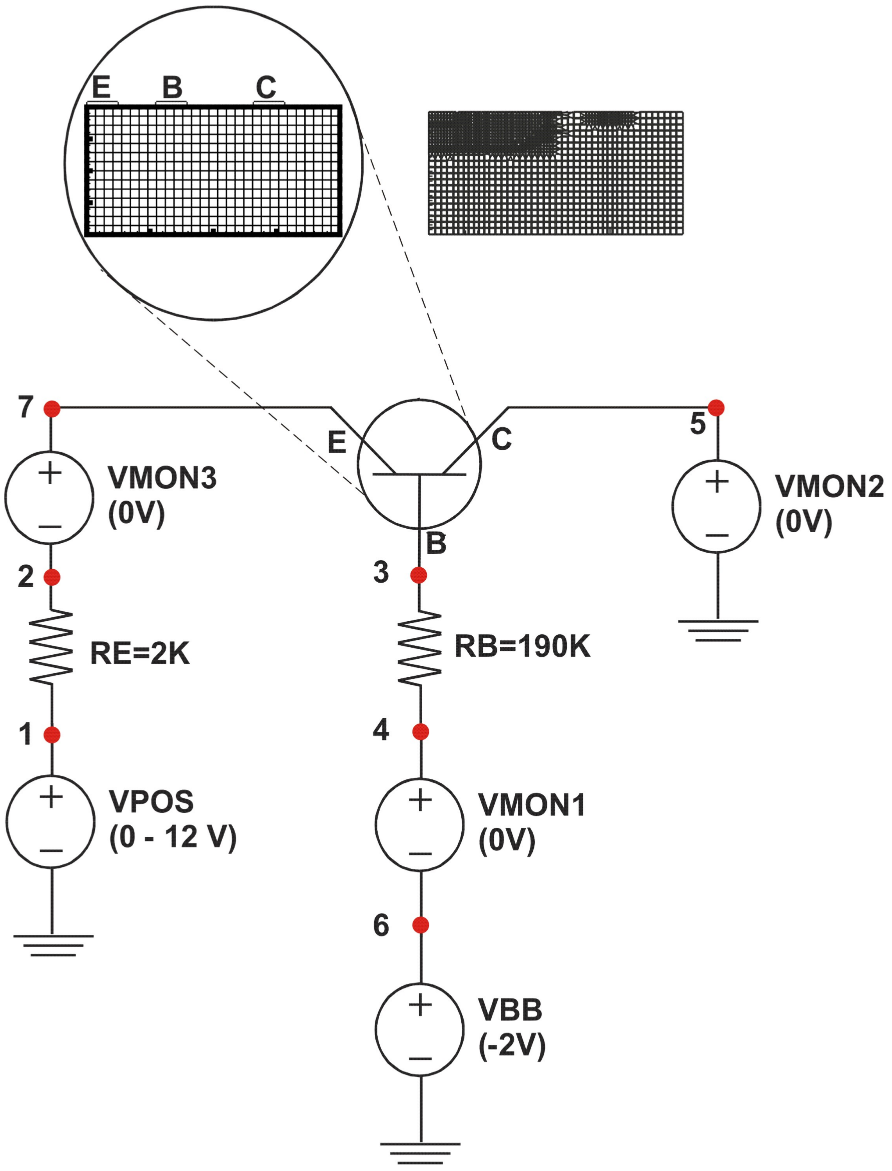


Figure 11.6. Two-Dimensional BJT Circuit Schematic. This schematic is for the circuit described by the netlist in Figure 11.5. The mesh in the large circle is the mesh used in the example. The other mesh, which contains some mesh refinement, is included in the figure as an example of what is possible with an external mesh generator.

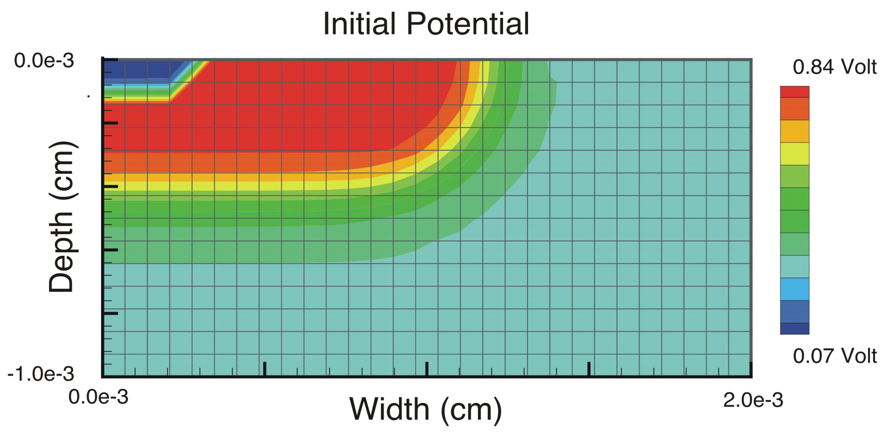


Figure 11.7. Initial Two-Dimensional BJT Result. Contour plot of the electrostatic potential at the first DC sweep step of the netlist in Figure 11.5. Note the mesh, which was generated using the internal "uniform mesh" option. This plot was generated using Tecplot.

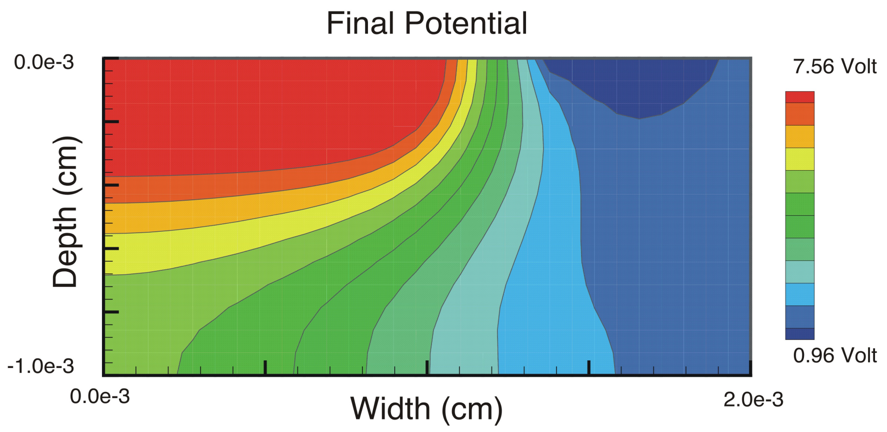


Figure 11.8. Final Two-Dimensional BJT Result. Contour plot of the electrostatic potential at the last DC sweep step of the netlist in Figure 11.5. This plot was generated using Tecplot.

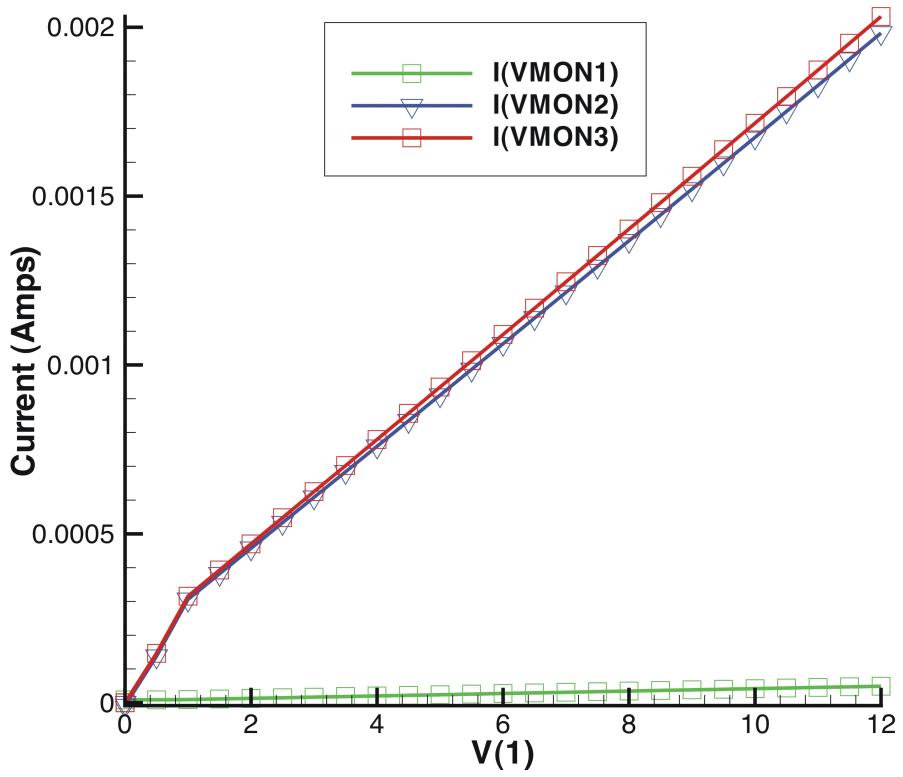


Figure 11.9. I-V Two-Dimensional BJT Result, for the netlist in Figure 11.5. The x-axis is in Volts. The three plotted currents are through the three BJT electrodes, and as expected they add (if corrected for sign) to zero. I(VMON1) is the base current, I(VMON2) the collector current, and I(VMON3) the emitter current. V(1) is the voltage applied to the emitter load resistor, RE. This plot was generated using Tecplot.

11.4 Doping Profile

In the two examples, no doping parameters were specified, and **Xyce** used the defaults. Default profiles are uniquely specified by the number of electrodes. In practice, especially for two-dimensional simulations, the user will generally need to specify the doping profile manually.

NOTE: If an external mesh (from the SGFramework) is used, the doping profile will be read in from the mesh file, and it is not necessary (or appropriate) to specify any doping in the netlist.

Manually Specifying the Doping

A circuit netlist, which includes a one-dimensional PDE device with a detailed, manual specification of the doping profile, is given in figure 11.10. A similar, two-dimensional, version of this problem is given in figure 11.12. For the purposes of this discussion, the one-dimensional example will be referred to, but information conveyed is equally applicable to the two-dimensional case.

In both examples, the parameters associated with doping are in red font. The doping is specified with one or more regions, which are summed together to get the total profile. Doping regions are specified in a tablular format, with each column representing a different region.

In the one-dimensional example, there are three regions, which are illustrated in figure 11.11. Region 1 is a uniform n-type doping, with a constant magnitude of 4.0e+12 donors per cubic cm. This magnitude is set by the parameter nmax. As the doping in this region is spatially uniform, the only meaningful parameters are `function` (which in this case specifies a spatially uniform distribution), `type` (ntype or ptype) and `nmax`. The others (nmin through flaty) are ignored for a spatially uniform region.

Region 2 is a more complicated region, in that the profile varies with space. This region is doped with p-type impurities, and has a Gaussian shape. Semiconductor processing often consists of an implant followed by an anneal, which results in a diffusive profile. The Gaussian function is a solution to the diffusion problem, when it is assumed that the impurity exists in a fixed quantity. Thus, the Gaussian shape is an appropriate choice for the doping regions of a lot of devices.

The peak of the Region 2 doping profile is given by the parameter nmin, and is 1.0e+19 acceptors per cubic cm. This peak has a location in the device which is specified by `xloc=24.5e-4 cm`. The parmeters nmin and xwidth are fitting parameters.

Region 3 is also based on a Gaussian function, but unlike Region 2, it is flat on one side of the peak. This is set by the `flatx` parameter. The "flat" parameters follow the convention given by Table 11.1.

```
Doping and Electrode specification example
TITLE     Xyce PN Junction Simulation
vscope   0   1   0.0
rscope   2   1   50.0
cid      3   0   1.0u
r1       4   3   1515.0
vid      4   0   5.00
Z1DIODE 2 3 PDEDIODE nx=301 l=26.0e-4
* DOPING REGIONS:    region 1,  region 2,   region 3
+ region= [function = uniform,  gaussian,  gaussian
+          type     =   ntype,     ptype,      ntype
+          nmax     = 4.0e+12,   1.0e+19,   1.0e+18
+          nmin     = 0.0e+00,   4.0e+12,   4.0e+12
+          xloc     =    0.0 ,  24.5e-04,   9.0e-04
+          xwidth   =    0.0 ,   4.5e-04,   8.0e-04
+          flatx    =    0   ,         0 ,       -1 ]
*--------end of  Diode PDE device ----------------
.MODEL PDEDIODE  ZOD  level=1
.options LINSOL ksparse=0
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END
```

Figure 11.10. One-dimensional example, with detailed doping.

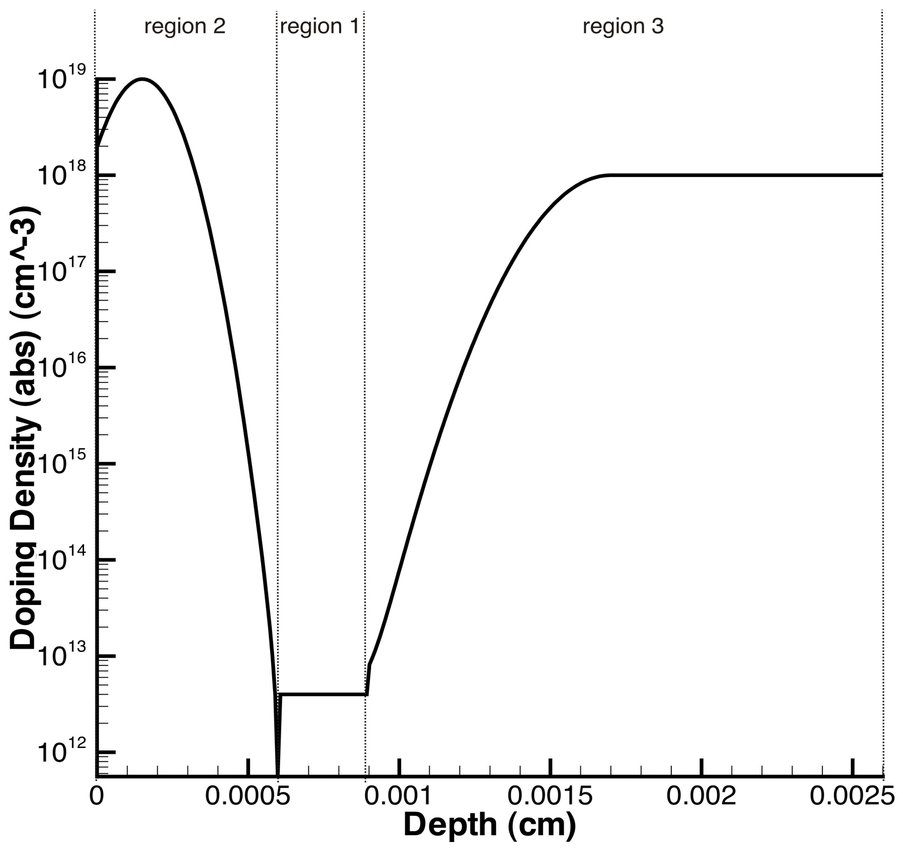


Figure 11.11. Doping Profile, Absolute Value. This corresponds to the doping specified by the netlist in figure 11.10

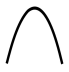
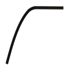
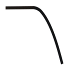
flatx or flaty value	Description	1D Cross Section
0	Gaussian on both sides of the peak (`xloc`) location.	
+1	Gaussian if `x>xloc`, flat (constant at the peak value) if `x<xloc`.	
-1	Gaussian if `x<xloc`, flat (constant at the peak value) if `x>xloc`.	

Table 11.1: Description of the flatx, flaty doping parameters

Default Doping Profiles

Xyce has a few default doping profiles which are invoked if the user doesn't bother to specify detailed doping information. The default doping profiles are an artifact of early PDE device development in **Xyce**, but are sometimes still useful. In particular, the simple step-junction diode is often a useful cannonical problem. It is convenient to invoke a step junction doping without having to use the more complex region tabular specification.

Most real devices will have doping profiles that do not exactly match the default profiles. When attempting to simulate a realistic device, it will be neccessary to skip the defaults and use the region tables described in the previous section.

One Dimensional Case

For the one-dimensional case, it is assumed that the doping profile is that of a simple junction diode, with the junction location exactly in the middle. The acceptor and donor concentrations are given by the parameters `Na` and `Nd`, respectively.

Note that the usage of `Na` and `Nd`, implicitly specifies a step junction doping profile, and is mutually exclusive with the more complex "doping region" table specification, described in section 11.4. If a netlist is input to **Xyce** which includes both a region table and `Na` (or `Nd`), the code will immediately exit with an error.

Two Dimensional Case

Doping level defaults in the two dimensional case are somewhat more complicated than in the one-dimensional case, because having two-dimensions allows for more configurations, and an arbitrary number (2-4) of electrodes. In **Xyce**, it was decided that the default doping profiles would be determined uniquely by the number of electrodes. The three available default dopings are given in Table 11.2. In the case of the BJT and MOSFET dopings, it is possible to specify either n-type or p-type using the `type` instance parameter. If the detailed, manual doping is used, then the `type` parameter is ignored.

For a two-electrode device, the default doping is that of a simple diode. The acceptor and donor doping parameters, `Na` and `Nd` are used in the same manner as in the one-dimensional device. As in the one-dimsensional device, the junction is assumed to be exactly in the middle of the domain.

For a three-electrode device (like the example), the default doping is that of a bipolar junction transistor (BJT). By default the transistor is a PNP, but by setting the instance parameter `type=NPN`, an NPN transistor can be specified instead. The two-dimensional example in section 11.3 relies on this default.

For a four-terminal device, the default doping is that of a metal-oxide-semiconductor (MOSFET). Currently, the maximum number of electrodes is four, and no default profiles are available for more than four electrodes. By default this transistor is assumed to be NMOS, rather than PMOS.

Number of Electrodes	**Doping Profile**
2	Step Function Diode
3	Bipolar Junction Transistor (BJT)
4	Metal-Oxide Semiconductor Field-Effect Transistor(MOSFET)

Table 11.2: Default Doping profiles for different numbers of electrodes

11.5 Electrodes

In the two examples, minimal electrode were specified, and **Xyce** used the defaults. In practice, especially for two-dimensional simulations, the user will need to specify the electrodes in more detail.

NOTE: If an external mesh (from the SGFramework) is used, some of the electrode information (the locations, and lengths) will be specified in the mesh file, so they should not be specified in the netlist.

Manually Specifying the Electrodes

A detailed electrode specification is specifed in blue font in Figure 11.12. As with the doping parameters, the electrode parameters are specified in a tabular format, in which each columns of the table specifies the parameters for a different electrode. The most important parameter (for getting the code to run without immediately exiting with an error) is the `name` parameter. It is the only required parameter.

The number of specified electrodes must match the number of connected circuit nodes, and the order of the electrode columns, from left to right, is in the same order as the circuit nodes, also from left to right. In the example of Figure 11.12, the first electrode column, which specifies an electrode named “anode”, is connected to the circuit through circuit node 2. Respectively, the second column, for the “cathode” electrode, is connected to the circuit by circuit node 3.

If using an external mesh (see section 11.6), the external mesh file must have this same number of electrodes as well. Also, if using the external mesh, the electrode names specifed in the electrode table must match (case insensitive) with the electrode names used by the external mesh.

Boundary Conditions

In the example, the `bc` parameter has been set to “Dirichlet” on all the electrodes, which is the default. The `bc` parameter sets the type of boundary condition that is applied to the density variables, the electron density and the hole density. There are two possible settings for the `bc` parameter, Dirichlet and Neumann. If Dirichlet is specified, the electron and hole densities are set to a specific value at the contact, and the applied values enforce charge neutrality. See the **Xyce** Reference Guide for the charge-neutral equation [2]. If Neumann is specified, a zero-flux condition is applied, which enforces that the current through the electrode will be zero.

This parameter does not affect the electrostatic potential boundary condition. The boundary condition applied to the potential is always Dirichlet, and is (in part) determined from

```
Doping and Electrode specification example
vscope    1   0   0.0
rscope    2   1   50.0
cid       3   0   1.0u
r1        4   3   1515.0
vid       4   0   1.00
*------------- Diode PDE device ------------------
Z1DIODE 2 3 PDEDIODE
+ tecplotlevel=1 txtdatalevel=1 cyl=1
+ meshfile=internal.msh
+ nx=25  l=70.0e-4 ny=40  w=26.0e-4
 * ELECTRODES:                ckt node 2, ckt node 3
+ node = [name            =     anode,    cathode
+         bc              = dirichlet,  dirichlet
+         start           =       0.0,        0.0
+         end             =   70.0e-4,    70.0e-4
+         side            =       top,     bottom
+         material        =   neutral,    neutral
+         oxideBndryFlag =          0,          0 ]
* DOPING REGIONS:    region 1,  region 2,   region 3
+ region= [function = uniform,  gaussian,  gaussian
+          type     =   ntype,     ptype,     ntype
+          nmax     = 4.0e+12,   1.0e+19,   1.0e+18
+          nmin     = 0.0e+00,   4.0e+12,   4.0e+12
+          xloc     =    0.0 ,  60.0e-04,    100.0
+          xwidth   =    0.0 ,   4.0e-04,      1.0
+          yloc     =    0.0 ,  24.5e-04,   9.0e-04
+          ywidth   =    0.0 ,   4.5e-04,   8.0e-04
+          flatx    =    0   ,       -1 ,       -1
+          flaty    =    0   ,        0 ,       -1 ]
*--------end of  Diode PDE device ----------------
.MODEL PDEDIODE  ZOD  level=2
.options LINSOL ksparse=0
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END
```

Figure 11.12. Two-dimensional example, with detailed doping and detailed electrodes.

the connected nodal voltage. To apply a specific voltage to an electrode contact, a voltage source should be attached to it, such as `VBB` in the schematic Figure 11.6.

Electrode Material

Several different electrode materials can be specified. A list is given in Table 11.3. The main effect of any metal (non-neutral) material is the impose a Schottky barrier at the contact. This generally makes numerical solution more difficult, so any materials should be applied with caution.

The **Xyce** Reference Guide [2] has a detailed description of Schottky barriers and how they are imposed on contacts in **Xyce**. Also, values for electron affinities of various bulk materials and workfunction values for the various metal contacts are given in the Reference Guide.

Material	Symbol	Comments
neutral	neutral	Default
aluminum	al	
p+-polysilicon	ppoly	
n+-polysilicon	npoly	
molybdenum	mo	
tungsten	w	
molybdenum disilicide	modi	
tungsten disilicide	wdi	
copper	cu	
platinum	pt	
gold	au	

Table 11.3: Electrode Material Options. Neutral contacts are the default, and pose the least problem to the solvers.

There is also an `oxideBndryFlag` parameter, which if set to true (`1`), will model the contact as having an oxide layer in between the metal contact and the bulk semiconductor. Note that this oxide layer model does not currently include displacement current, so transient capacitive effects will not be seen in the results.

Location Parameters

Each electrode has three location parameters: `start`, `end`, and `side`. These are only necessary if using the internal mesh and should not be specified if using an external, SGFramework mesh.

For the internal mesh, the mesh is assumed to be rectangular, and any electrode is assumed to be on one of the four sides. The four side possibilities are: `top`, `bottom`, `right` and `left`. These four sides are parallel to mesh directions. The `start` and `end` parameters are floating point numbers which specify the starting and ending location of an electrode, in units of cm.

The lower left hand corner of the mesh rectangle is located at the origin. A `side=bottom` electrode with `start=0.0` and `end=1.0e-4` will originate at the lower left hand corner of the mesh (x=0.0, y=0.0) and end at (x=1.0e-4, y=0.0).

NOTE: Xyce will attempt to match the specified electrode to the specified mesh. However, if the user specifies a mesh that is not consistent with the electrode locations, the electrodes will not be able to have the exact length specified. For example, if the mesh spacing is $\Delta x =$ 1.0e-5, then the electrodes can only have a length that is a multiple of 1.0e-5.

Electrode Defaults

There are defaults for all the electrode parameters except the names. In practice, the locations of the electrodes will usually be explicitly specified (either using the electrode table, or as part of an external mesh file). Default electrode locations have been created to correspond with the default dopings, and they should only be used in that context.

Location Parameters

In practice, the locations of the electrodes will usually be explicitly specified, but they have defaults to correspond with the default dopings. The default electrode locations in one-dimensional devices are for that of a diode. One electrode is located at `x=xmin`, while the other is located at `x=xmax`.

The default electrode locations in two-dimensional devices are dependent on the number of electrodes, similar to the default dopings. Table 11.2 can be used to determine the configurations. For the two-terminal diode, the two electrodes are along the y-axis, at the `x=xmin` and `x=xmax` extrema. For the three-terminal BJT , all three electrodes are parallel to the x-axis, along the top, at `y=ymax`. For the four-terminal MOSFET, the drain, gate, and source electrodes are also along the top, but the bulk electrode spans the entire length of the bottom of the mesh, at `y=ymin`.

Other Parameters

The default contact material is `neutral`. The default `oxideBndryFlag` is false (`0`). The default boundary condition (`bc`) is `Dirichlet`.

11.6 Meshes

Meshes from the SG Framework (External, 2D)

It is possible to have **Xyce** read in a two-dimensional mesh which was generated externally, by the SGFramework [12]. The mesh pictured in Figure 11.1 is such a mesh, and so is the refined mesh (not inside the circle) in Figure 11.6. To use an SGF-generated mesh, the instance parameter, ”meshfile” must be used, and set to be the name of the SGFramework-generated file. **Xyce** will assume that the mesh file is located in the local execution directory. One advantage of using an externally generated mesh (over an internally generated mesh - see next section) is that external meshing tools are more sophisticated, and in particular have mesh refinement capabilities.

Instructions for the usage of the SGFramework is outside the scope of this document. If the user wishes to generate meshes in this manner, it is best to consult Kramer [12]. Future versions of **Xyce** may accept mesh files generated by other mesh generators, such as Cubit [15].

Cartesian Meshes (Internal, 1D and 2D)

One dimensional and two-dimensional devices can both create cartesian meshes, without requiring an external mesh generator. For the two-dimensional devices, it is necessary to specify `meshfile=internal.msh` to invoke the cartesian meshing capability. For one-dimensional devices, this isn't needed, as there is no other option.

Meshes generated in this manner are very simple, in that there are only two parameters per dimension, and the resulting mesh is uniform. An example of such a mesh can be seen in Figure 11.7. The mesh spacing is determined from the following expressions:

$$\Delta x = \frac{l}{nx - 1} \tag{11.8}$$

$$\Delta y = \frac{w}{ny - 1} \tag{11.9}$$

This mesh specification assumes that the domain is a rectangle. Non-rectangular domains can only be described using an external mesh program.

Cylindrical meshes, 2D

For two-dimensional devices, the simulation area may be a cylinder slice. This capability is turned on by the instance parameter, `cyl=1`. For an example, see Figure 11.13. It is assumed that the axis of the cylinder corresponds to the minium radius (or x-axis value) of the mesh, while the circumference corresponds to the maximum radius (or maximum x-axis value). This feature can be applied to either external or internal two-dimensional meshes.

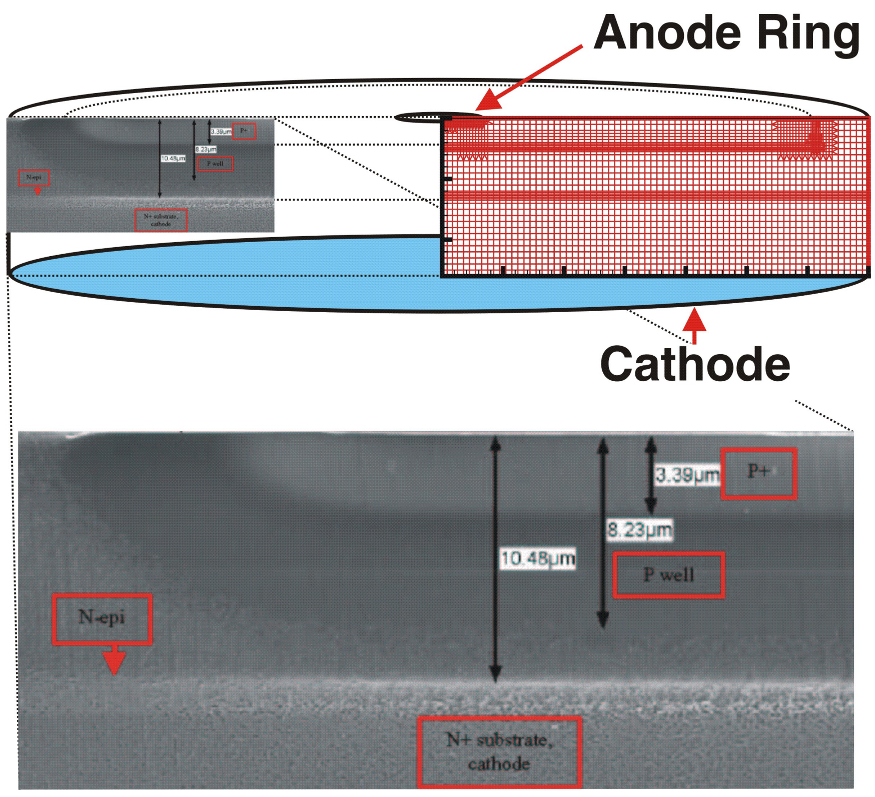


Figure 11.13. Cylindrical Mesh Example. This mesh has been designed to match the electron microsope image, which is of a stockpile device.

11.7 Mobility Models

There are several mobility models available to both the one and two dimensional devices, and they are listed in Table 11.4. These models are fairly common, and can be found in most device simulators. [10] [11] These models are described in more detail in the **Xyce** Reference Guide [2].

Mobility Name	Description	Reference
arora	Basic mobility model	Arora, et al. [16]
analytic	Basic mobility model	Caughy and Thomas [17]
carr	Includes carrier-carrier interactions	Dorkel and Leturq [18]

Table 11.4: Mobility models available for PDE devices

Specifying the mobility model from the netlist is done by setting the `mobmodel` parameter to the name of the model. Model names are given in the first column of Table 11.4. The mobility model is specified as an instance parameter on the PDE device instance line, as (typically) `mobmodel=arora`. See the usage in Figure 11.5 for a more detailed example.

The default mobility is the ”carr” mobility, which includes carrier-carrier interactions. This model has a stronger dependence on carrier density than the other two models, and introduces some nonlinearity into the problem. If having convergence problems, consider using either the ”arora” or ”analytic” model, as both of these models are a little bit simpler.

11.8 Bulk Materials

The bulk material is specifed using the `bulkmaterial` instance parameter. **Xyce** currently supports Silicon (`si`) as a bulk material and this is the default. It can also simulate Gallium Arsenide (`gaas`) and Germanium (`ge`), but these materials have not been extensively tested.

The mobility models described in the previous section each support all three materials, and the dielectric permittivity is correct for all three, but the carrier lifetime models may not be. These issues will be resolved in a future **Xyce** release.

11.9 Solver Options

Problems that are based on PDE devices have different optimal solver settings than do analog circuit problems. Generally, as PDE devices are mesh-based, and have a more predictable topology, iterative linear solvers have a better chance of being successful than they do for analog circuit simulation. On the nonlinear solver level, voltage limiting doesn't have an obvious application to PDE devices, and quadratic line search appears to be the best algorithm. The solver options specified in the example netlist Figure 11.5 are adequate for simulations that have a simple (linear) circuit attached.

For problems which involve a complicated external circuit, it is best to apply the two-level Newton algorithm to the nonlinear solve. This algorithm is described in detail in Keiter [19] and Mayaram [20]. While this algorithm has been implemented and exercised within **Xyce**, it is not part of the Release 2.0 version. To use this algorithm, the user will need to obtain a development branch build of **Xyce**, or wait until Release 2.1.

11.10 Output and Visualization

Using the `.PRINT` Command

For simple plots (such as I-V curves), output results for **Xyce** can be generated with the `.PRINT` statement, which is described in detail in section 9.1. Figures 11.4 and 11.9 are examples of the kind of data that is produced with `.PRINT` statement netlist commands. These particular figures were plotted in Tecplot, but many other plotting programs would also have worked, including XDAMP [21].

Multi-dimensional Plots

Device simulation has visualization needs which go beyond that of conventional circuit simulation. Multi-dimensional perspective and/or contour plots are often desirable. **Xyce** is capabable of outputting multi-dimensional plot data in several formats, including Tecplot, GnuPlot, and sgplot. Currently, the options for each of these formats can only enable or disable the output of files, and when enabled, a new file (or a new append to an existing file) will happen at every time step or DC sweep step. For long simulations, this may produce a prohibitive number of files. Currently, there is no equivalent to the `.OPTIONS OUTPUT INITIAL_INTERVAL` command, nor does the output of plot data currently use this command. Plot files are either output at every step or not at all.

For each type of plot file, the file is placed in the execution directory. Each individual device instance is given a unique file, or files, and the file names are derived from the name of the PDE device instance. The instance names provides the prefix, and the file type (tecplot, gnuplot, sgplot) determines the suffix.

Tecplot Data

Tecplot is a commercial plotting program from Amtec Engineering, Inc., and is the best choice for creating contour plots of spatially dependent data. All of the graphical examples in this chapter were created with Tecplot. (see Figures 11.7 and 11.8 for examples) The output of Tecplot files is enabled using the instance parameter, `tecplotlevel=1`. If set to zero, no tecplot files are output. If set to one, a separate tecplot file is output for each nonlinear solve. If set to two, a single tecplot file, which contains data for every nonlinear solve is created and is appended at the end of each solve.

By default `tecplotlevel` is set to one, meaning the code will, by default produce a separate Tecplot file for each nonlinear solve. The suffix for Tecplot data files is *.dat. Internally, the file is an ASCII text file. Tecplot does have a binary format, but **Xyce** has not yet been set up to use it.

Note that it is also possible to set `tecplotlevel=2`. Doing this will force **Xyce** to create one single tecplot file, and the data from each solve will be appended to this file as a separate zone. This makes it possible to use Tecplot to create annimations.

Gnuplot Data

Gnuplot is an open source plotting program, which is available on most Linux/Unix platforms. The parameter for this type of output is `gnuplotlevel=1`. This type of output file is off (zero) by default, meaning no gnuplot files will be output. The suffix for Gnuplot files is *Gnu.dat. Like tecplot files, Gnuplot files are also in ASCII text format.

NOTE: Gnuplot will only work with structured Cartesian meshes. Externally created, unstructured meshes (even ones that appear Cartesian) cannot be plotted with Gnuplot.

Sgplot Data

Sgplot is the plotting program for the SGFramework [12]. The parameter for this type of output is `sgplotlevel=1`. This type of output file is off (zero) by default. The suffix for Sgplot data files is *.res. Interally this file is in binary format. Note that it is not a machine-independent file format.

Volume Averaged Data

Xyce can also output volume-averaged information for each PDE device. This is enabled by setting the instance parameter, `txtdatalevel=1`. It is off (zero) by default, meaning no text files with volume averaged data will be output.

This page is left intentionally blank

Bibliography

[1] M. S. Eldred, A. A. Giunta, B. G. van Bloemen Waanders, S. F. Wojtkiewicz Jr., W. E. Hart, and M. P. Alleva. DAKOTA, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Version 3.0 Reference Manual. Technical Report SAND2001-3796, Sandia National Laboratories, Albuquerque, NM, April 2002.

[2] Scott A. Hutchinson, Eric R. Keiter, Robert J. Hoekstra, Lon J. Waters, Thomas V. Russo, Eric L. Rankin, Roger P. Pawlowski, and Steven D. Wix. Xyce parallel electronic simulator: Reference guide, version 2.0. Technical Report SAND2004-xxxx, Sandia National Laboratories, Albuquerque, NM, June 2004.

[3] Orcad PSpice User's Guide. Technical report, Orcad, Inc., 1998.

[4] *ChileCAD Schematic Capture Tool*. `http://www.sandia.gov/mstc/products/elect-comp/chileca`
.

[5] A. S. Grove. *Physics and Technology of Semiconductor Devices*. John Wiley and Sons, Inc., 1967.

[6] H. A. Watts, E. R. Keiter, S. A. Hutchinson, and R. J. Hoekstra. Time integration for the Xyce parallel electronic simulator. In *ISCAS 01*, October 2000.

[7] J. Roychowdhury. *Private Communication*, 2003.

[8] Bruce Hendrickson and Robert Leland. The Chaco User's Guide: Version 2.0. Technical Report SAND94–2692, Sandia National Laboratories, Albuquerque, NM, December 1994.

[9] Erik Boman, Karen Device, Robert Heaphy, Bruce Hendrickson, William F. Mitchell, Matthew St. John, and Courtenay Vaughan. *Zoltan: Data-Management Services for Parallel Applications:User's Guide*. `http://www.cs.sandia.gov/Zoltan/Zoltan.html`, 2004.

[10] Z. Yu, D. CHen, L. So, , and R. W. Dutton. Pisces-2et–two dimensional device simulation for silicon and heterostructures. Technical report, Stanford University, 1994.

[11] Davinci User's Manual. Technical report, TCAD Business Unit, Avanti! Corporation, 1998.

[12] Kevin M. Kramer and W. Nicholas G. Hitchon. *Semiconductor Devices: A Simulation Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[13] S. Selberherr. *Analysis and Simulation of Semiconductor Devices*. Springer-Verlag, New York, 1984.

[14] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Eric L. Rankin, Thomas V. Russo, and Lon J. Waters. Computational algorithms for device-circuit coupling. Technical Report SAND2003-0080, Sandia National Laboratories, Albuquerque, NM, January 2003.

[15] *CUBIT Mesh Generation Toolsuite*. `http://sass1693.sandia.gov/cubit/`.

[16] N.D. Arora, J.R. Hauser, and D.J. Roulston. Electron and hole mobilities in silicon as a function of concentration and temperature. *IEEE Transactions on Electron Devices*, ED-29:292–295, 1982.

[17] D.M. Caughey and R.E. Thomas. Carrier mobilities in silicon empirically related to doping and field. *Proc. IEEE*, 55:2192–2193, 1967.

[18] J.M. Dorkel and Ph. Leturq. Carrier mobilities in silicon semi-empirically related to temperature, doping, and injection level. *Solid-State Electronics*, 24(9):821–825, 1981.

[19] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Eric L. Rankin, Thomas V. Russo, and Lon J. Waters. Computational algorithms for device-circuit coupling. Technical Report SAND2003-0080, Sandia National Laboratories, Albuquerque, NM, January 2003.

[20] Kartikeya Mayaram and Donald O. Pederson. Coupling algorithms for mixed-level circuit and device simulation. *IEEE Transactions on Computer Aided Design*, II(8):1003–1012, 1992.

[21] *XDAMP Graphical User Interface*. `http://www.cs.sandia.gov/xyce/xdamp.html` .

Index

Xyce
running, 23
running in parallel, 25, 81
.DC, 31
.INCLUDE, 51
.MODEL, 40
.OP, 58
.OPTIONS
LINSOL, 84, 85
OUTPUT, 63, 78
PARALLEL, 84, 85
RESTART, 62, 63
.PRINT, 78, 79
DC, 31
FORMAT, 80
TRAN, 34, 78
.STEP, 65
.SUBCKT, 40
.TRAN, 34, 61
runxyce, 22, 23
runxyce.bat, 22, 23
xmpirun, 22, 23

analog behavioral modeling (ABM), 44, 53
analysis
DC, 17, 58
DC sweep, 31
DC sweep, 58
STEP, 65
transient, 17, 33, 61

behavioral model, 4, 41, 44
analog behavioral modeling (ABM), 44, 53, 54
lookup table, 55
bias point, 58, 61
bifurcation, 71

Chaco, 83, 84
checkpoint, 62
format, 63
ChileSPICE, 19
circuit
elements, 38
simulation, 38
topology, 38, 39
command line, 23, 25
arguments, 24
output, 24
continuation, 71

DAKOTA, 17
DC analysis, 58
DC Sweep, 58
DC sweep, 31
OP Analysis, 58
running, 58
device
B (nonlinear dependent) source, 54
analog, 40, 41
analog device summary, 42
B source, 41
behavioral, 54
behavioral model, 16, 17, 41
bipolar junction transistor (BJT, 41
capacitor, 41
device types, 41
diode, 41
independent current source, 41
independent voltage source, 41
inductor, 41
instance, 41
MOSFET, 42
mutual inductor, 41

nonlinear dependent source, 41
package, 17
PDE Devices, 42
resistor, 41
specifying ABM devices, 54
subcircuit, 42
transmission line, 42
voltage controlled current source, 41
voltage controlled switch, 42
voltage controlled voltage source, 41

Example
checkpointing, 63
circuit construction, 28
DC sweep, 31
declaring parameters, 43
restarting, 63
subcircuit model definition, 49, 50
transient analysis, 33
using expressions, 44
using parameters, 43
expressions, 43
additional constructs for ABM modeling, 55
arithmetic, 46
example, 44
lookup table, 55
operators, 45
time-dependent, 55
using, 44
valid constructs, 44

graph partitioning, 83
ground nodes, 39
GUI, 23

homotopy, 71, 72

Microsoft Windows, 23
model
definition, 48
model organization, 51
MPI, 23

netlist, 28, 38
.END, 38
.END statement, 29
analog devices, 40
arithmetic expressions, 46
command elements, 40
comments, 29, 39
device description, 40
element, 38
end line, 39
expression operators, 45
expressions, 43
first line special, 39
in-line comments, 39
model definition, 40
node names, 39
parameters, 42
restart, 62
scaling factors, 38
sources, 61
subcircuit, 40
title, 29
title line, 38, 39
using expressions, 44
node names, 39

OP analysis, 58
output
comma separated value, 23
log file, 23
specifying file name, 23
time values, 62

parallel
communication, 83
computing, 15, 16
distributed-memory, 16
efficiency, 16, 17
graph partitioning, 83
large scale, 16
load balance, 83
message passing, 16
MPI, 23
number of processors, 24
shared-memory, 16
parallelGuidance, 26
parameter
declaring, 43
using in expressions, 43

PDE Device Modeling, 87
PDE Devices, 42
platforms
  Apple/OSX, 24
  HP/Compaq/Tru64, 24
  Intel X86/FreeBSD, 24
  Intel X86/Linux, 24
  Intel X86/Microsoft Windows 2000, 24
  SGI/IRIX, 24
PSpice, 19, 28
  Probe, 80

Reference Guide, 19
restart, 62, 63
  format, 63
  limitations in parallel, 84
results
  evaluating, 80
  output control, 78
  output frequency, 78
  output options, 77
  print commands, 79
running **Xyce**, 23
runxyce, 23

Sandia National Laboratories, 15
schematic capture, 28
simulation
  analog, 17
  device, 17
  digital, 17
  mixed signal, 17
solvers
  iterative linear, 84
  transient, 62
sources, 61
  defining time-dependent, 61
  time-dependent, 61
  waveforms, 62
SPICE, 28, 38
STEP parametric analysis, 65
subcircuit
  hierarchy, 49
  scope, 50
Synopsys
  DaVinci, 18
  Medici, 18

time step
  maximum size, 62
  size, 62
topology, 39
transient analysis, 33, 61

Unix, 19
Users of other circuit codes, 19

Windows, 23
  Command Prompt, 23

xmpirun, 23

ZOLTAN, 83, 84

DISTRIBUTION:

1 Steven P. Castillo
Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 3-o
Las Cruces, NM 88003

1 Kwong T. Ng
Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 3-o
Las Cruces, NM 88003

1 Nick Hitchon
Electrical and Computer Engineering
University of Wisconsin
1415 Engineering Drive
Madison, WI 53706

1 Mark Kushner
Department of Electrical and Computer Engineering
University of Illinois
1406 W. Green Street
Urbana, IL 61801

1 Ron Kielkowski
RCG Research, Inc
8605 Allisonville Rd, Suite 370
Indianapolis, In 46250

1 Mike Davis
Software Federation, Inc.
211 Highview Drive
Boulder, Co 80304

1 Wendland Beezhold
Idaho Accelerator Center
1500 Alvin Ricken Drive
Pocatello, Idaho 83201

1 Kartikeya Mayaram
Department of Electrical and Computer Engineering
Oregon State University
Corvallis, OR 97331-3211

1 Linda Petzold
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106-5070

1 Jaijeet Roychowdhury
4-174 EE/CSci Building
200 Union Street S.E.
University of Minnesota
Minneapolis, MN 55455

1 C.-J. Richard Shi
VLSI and Electronic Design Automation
210 EE/CSE Bldg.
Box 352500
University of Washington
Seattle, WA 98195

1 Homer F. Walker
WPI Mathematical Sciences
100 Institute Road
Worcester, MA 01609

1 Dan Yergeau
CISX 334
Via Ortega
Stanford, CA 94305-4075

1 Masha Sosonkina
319 Heller Hall
10 University Dr.
Duluth, MN 55812

1 Misha Elena Kilmer
113 Bromfield-Pearson Bldg.
Tufts University
Medford, MA 02155

1 Tim Davis
P.O. Box 116120
University of Florida
Gainesville, FL 32611-6120

1 Achim Basermann
C&C Research Laboratories, NEC Europe Ltd.
Rathausallee 10
D-53757 Sankt Augustin
Germany

1 Philip A. Wilsey
Clifton Labs
7450 Montgomery Road
Suite 300
Cincinnati, Ohio 45236

1 Dale E. Martin
Clifton Labs
7450 Montgomery Road
Suite 300
Cincinnati, Ohio 45236

1 Lon Waters
CoMet Solutions, Inc.
11811 Menaul Blvd NE
Suite No. 1
Albuquerque, NM 87112

1 MS 0151
Tom Hunter, 09000

1 MS 0513
Al Romig, 01000

1 MS 0457
John Stichman, 02000

1 MS 0321
Bill Camp, 09200

1 MS 0841
Thomas C. Bickel, 09100

1 MS 1079
Marion Scott, 01700

1 MS 9003
Kenneth E. Washington, 08900

1 MS 0318
Paul Yarrington, 09230

1 MS 1071
Mike Knoll, 01730

1 MS 0310
Robert Leland, 09220

1 MS 0316
Sudip Dosanjh, 09233

1 MS 0525
Paul V. Plunkett, 01734

1 MS 0835
Steven N. Kempka, 09141

1 MS 0826
John D. Zepper, 09143

1 MS 0824
Jaime L. Moya, 09130

1 MS 0828
Martin Pilch, 09133

1 MS 0828
Anthony A. Giunta, 09133

1 MS 0139
Stephen E. Lott, 09905

1 MS 0310
Mark D. Rintoul, 09212

1 MS 1110
David Womble, 09214

1 MS 1111
Bruce Hendrickson, 09215

1 MS 1110
Neil Pundit, 09223

1 MS 1110
Doug Doerfler, 09224

1 MS 0822
Philip Heermann, 09227

1 MS 0819
Edward Boucheron, 09231

1 MS 0820
Patrick Chavez, 09232

1 MS 0316
John Aidun, 09235

1 MS 0316
Scott A. Hutchinson, 09233

10 MS 0316
Eric R. Keiter, 09233

1 MS 0316
Deborah Fixel, 09233

1 MS 0316
Robert J. Hoekstra, 09233

1 MS 0316
Joseph P. Castro, 09233

1 MS 0316
David R. Gardner, 09233

1 MS 0316
Gary Hennigan, 09233

1 MS 0316
Roger Pawlowski, 09233

1 MS 0316
Richard Schiek, 09233

1 MS 1111
John N. Shadid, 09233

1 MS 1111
Andrew Salinger, 09233

1 MS 0316
Paul Lin, 09233

1 MS 0316
Siriphone C. Kuthakun, 09233

1 MS 0807
David N. Shirley, 9328

1 MS 0807
Philip M. Campbell, 9328

1 MS 0847
Scott Mitchell, 09211

1 MS 0847
Mike Eldred, 09211

1 MS 0847
Tim Trucano, 09211

1 MS 0847
Bart van Bloemen Waanders, 09211

1 MS 0196
Elebeoba May, 09212

1 MS 1110
Todd Coffey, 09214

1 MS 1110
David Day, 09214

1 MS 1110
Mike Heroux, 09214

1 MS 1110
James Willenbring, 09214

1 MS 1111
Karen Devine, 09215

1 MS 0310
Jim Ang, 09220

1 MS 1109
Robert Benner, 09224

1 MS 0822
Pat Crossno, 09227

1 MS 0822
David Rogers, 09227

1 MS 0316
Harry Hjalmarson, 09235

1 MS 0525
Steven D. Wix, 01734

1 MS 0525
Thomas V. Russo, 01734

1 MS 0525
Regina Schells, 01734

1 MS 0525
Carolyn Bogdan, 01734

1 MS 0525
Mike Deveney, 01734

1 MS 0525
Raymond B. Heath, 01734

1 MS 0525
Ronald Sikorksi, 01734

1 MS 0525
Albert Nunez, 01734

1 MS 1081
Paul E. Dodd, 01762

1 MS 0660
Roger F. Billau, 09519

1 MS 0874
Robert Brocato, 01751

1 MS 1081
Charles E. Hembree, 01739

1 MS 0889
Neil R. Sorenson, 01832

1 MS 0311
Greg Lyons, 02616

1 MS 0311
Martin Stevenson, 02616

1 MS 0328
Fred Anderson, 02612

1 MS 0537
Perry Molley, 02331

1 MS 0537
Siviengxay Limary, 02331

1 MS 0537
John Dye, 02331

1 MS 0537
Barbara Wampler, 02331

1 MS 0537
Doug Weiss, 02333

1 MS 0537
Scott Holswade, 02333

1 MS 0537
George R. Laguna, 02333

1 MS 0405
Todd R. Jones, 12333

1 MS 0405
Thomas D. Brown, 12333

1 MS 0405
Donald C. Evans, 12333

1 MS 0405
Matthew T. Kerschen, 12333

1 MS 9101
Rex Eastin, 08232

1 MS 9101
Seung Choi, 08235

1 MS 9409
William P. Ballard, 08730

1 MS 9202
Kathryn R. Hughes, 08205

1 MS 9202
Rene L. Bierbaum, 08205

1 MS 9202
Kenneth D. Marx, 08205

1 MS 9202
Stephen L. Brandon, 08205

1 MS 9202
Jason Dimkoff, 08205

1 MS 9202
Brian E. Owens, 08205

1 MS 9401
Donna J. O'Connell, 08751

1 MS 9217
Stephen W. Thomas, 08950

1 MS 9217
Tamara G. Kolda, 08950

1 MS 9217
Kevin R. Long, 08950

1 MS 9915
Mitchel W. Sukalski, 08961

1 MS 1153
Larry D. Bacon, 15333

1 MS 1179
Leonard Lorence, 15341

1 MS 1179
David E. Beutler, 15341

1 MS 1179
Brian Franke, 15341

1 MS 0835
Randy Lorber, 09141

1 MS 1152
Mark L. Kiefer, 01642

1 MS 1137
Greg D. Valdez, 06224

1 MS 1137
Mark A. Gonzales, 06224

1 MS 1138
Rebecca Arnold, 06223

1 MS 1138
Charles Michael Williamson, 06223

1 MS 1138
Harvey C. Ogden, 06223

1 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 9616